Universidad Carlos III de Madrid
Escuela Politécnica Superior

SIMULATION OF A QUADROTOR UNMANNED AERIAL VEHICLE

A bachelor thesis submitted in fulfillment of the requirements for the degree in Aerospace Engineering

Academic Year 2015/2016

Author: Rubén Vega Astorga
Tutor: David Morante
Director: Manuel Sansurjo

Abstract

The main goal of this thesis is to validate a simulator model for a quadcopter manual and autonomous flight. It is intended to use the validated simulator for predicting quadcopter behaviour in future complex missions without the necessity of realizing real tests.

Characterization of the vehicle physical parameters for obtaining its main performance coefficients and its moments of inertia are needed for taking into account the quadcopter properties in the simulator. Therefore, the vehicle has been assembled in order to ensure the best characterization for each component. Then, a simulation environment for the quadcopter testing has been chosen.

SITL software has been selected due to its ability for simulating all the functions of the autopilot board used by the quadcopter. The work operation of this software has been analyzed, realizing that the equations of motions model used could be improved, since it was not included all the physical theoretical effects. Thus, the work analysis has been consisted in testing real and simulated flights and studying the influence of the improvent of the quadcopter equations of motion. Either manually controlled and autonomous missions tests have been done, in order to cover all the vehicle flight possibilities.

Based on the numerical results, it is shown how the equation of motion improvement are responsible of more realistic quadcopter flight simulations.

Acknowledgements

First of all I would like to thank my family, who have always support me in all the projects I have undertaken.

Secondly, I would like to express my gratitude to David Morante, who has been my ideal thesis supervisor. His sage advice, knowledge and support have greatly help me during all the project work.

Thirdly, I would like to thank Manuel Sanjurjo aiming me to undertake a project in such emerging field and giving me academical support every time I needed.

Finally, I would like to thank Manuel Soler for providing me technical support; and to Alvaro Melgosa Pascual, for encouraging and helping me during all the project.

Contents

List of Acronyms

$AESA$: Agencia Estatal de Seguridad Aérea

APM: ArduPilot Mega

ESC: Electronic Speed Controller

GCS: Ground Control Station

GPS: Global Position System

GUI: Graphical User Interface

IMU: Inertial Measurement Unit

PWM: Pulse Wide Modulation

RTL: Return To Land

$SITL$: Software In The Loop

TCP: Transmission Control Protocol

$MTOW$: Maximum Take-Off Weight

UAV: Unmanned Aerial Vehicle

UDP: User Data Protocol

List of Figures

List of Tables

Notation

ϕ: Roll Euler angle

θ: Pitch Euler angle

ψ: Yaw Euler angle

Ω_r: Propeller rotational velocity

A: Cross-section area of the blade

C_A: Quadcopter drag coefficient

C_D: Drag coefficient

C_T: Thrust coefficient

g: Gravity acceleration constant

i_B, j_B, k_B: Body axis reference frame

i_E, j_E, k_E: Earth axis reference frame

I_{xx}, I_{yy}, I_{zz}: Moments of inertia

J: Identity matrix

J_r: Blades moment of inertia

l: Arm length

m_0: Board and battery mass

m_{arm}: Quadcopter arm mass

m_b: Blades mass

m_q *or* m: Total quadcopter mass

M_j: Torque

Π_{hov}: Hover throttle

Q_i: Drag of blade i

R: Blades radius

R_E^B: Transformation matrix from Earth to Body axis

S: Reference quadcopter surface

T_{max}: Maximum thrust force

v_B: Quadcopter absolute velocity in body axis

v_t: Terminal velocity

v_B: Quadcopter rotational velocity in body axis

ω_t: Rotational terminal velocity

Chapter 1

Introduction

Unmmaned aerial vehicles are a relatively new technology that has spread its applications recently. In this chapter, the development of these vehicles and their importance nowadays are presented. Also, it is going to be introduced the goals of this project, together with its planning in time for achieving them. Finally, a general overview of this thesis structure is going to be explained, in order to help the reader with the the understanding of the project.

1.1 History

Unmanned aerial vehicles (UAV) have always been distinguished by having attributes very different in comparison the rest of aerial vehicles. One of them is their capability to be potentially smaller than any other vehicle with the same target mission, since they do not need to carry pilots on board. Because of this fact, they have been considered throughout the history as an excellent opportunity to develop higher versatility and greater performance vehicles, being the military and defence industries their principal investors.

The first UAV in history was launched by Samuel Pierpont Langley in 1896 [1]. It consisted in an unmanned flight of a steam-powered aircraft over the Potomac River. Nevertheless, no guidance navigation or control system was used. Improvements in this type of vehicles came soon with the beginning of the First World War. During this period, ranges of 40 miles were attained. The development of these aircraft continued during the Second World War. Remote control via radio frequency transmissions was introduced making the control systems more precise. However, it was some decades later when the final important advance in UAV arrived. The first autonomous

preprogrammed mission was flight by the Mach 3 ramjet-powered UAV. New developments continued appearing in the following decades with war as the main project stimulus, being also these ones, as the Vietnam War, the best scenary to prove the technological advances achieved. Lastly, recent technical innovations such as GPS (Global Position System), digital data links and satellite communications have been used in UAV, promoting the expansion of their application competences. On purpose, this technology has been considered in the last years as a great chance to oppose terrorism.

Nonetheless, despite UAV technology has been linked throughout the history to military purposes, civil applications have recently experienced growth in interest and importance. They range from terrain inspection and security purposes to leisure ones. UAV use has been spread for mines inspections and for gas and petroleum exploration and production due to their capacity for safely terrain exploration and for arriving places where other type of vehicles cannot be driven. Furthermore, other industrial sectors have been benefited by this technology. It can specially be appreciated in the agricultural sector in Spain, where UAVs are being used for detecting potential dangers for crops, taking advantage of the drone feature for ecologically exploring vast lands [3].

1.2 Socioeconomic framework

The unstoppable blow up of the UAV applications has led to industries to invest in this technology. Moreover, the eco-friendly possibilities of these products have contributed for these industries to gain the sympathy of the public.

Looking at the opinion of business experts, the UAV market is considered a great commercial opportunity, since this market is expected to grow in the following years, especially in the civil side (a growth of 19% versus a 5% in the military side). Furthermore, these conclusions are also valid for new firms in this sector, owing to UAV industry is still young and legacy military drone manufacturer clients does not interfere the civilian sector.

Finally, regarding the geographic zones where it is being experienced the UAV market to growth, it is remarkable that many of the new drone manufactures are taking place outside USA, the traditional UAV developer. According to BI studies, Canada with the Aeryon firm, China through DJI firm and Korea with Gryphon firm are examples of firm growth in this market.

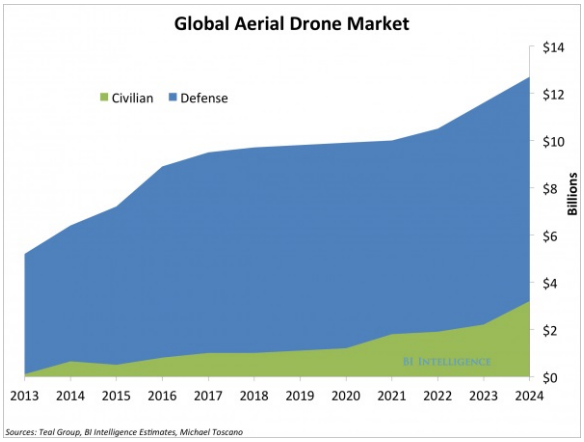

Figure 1.1: Global Aerial Drone Market expected evolution. BI Business Insider analysis [4]

1.3 Legal framework

The appearance of UAV for civil applications is still very recent, so that the legal framework is in continuous development. In fact, it will be necessary to gather these laws for all the countries in a near future. According to the Spanish legislation [5], the actual law regarding UAV is the following:

- For aircraft with a Maximum Take-Off Weight (MTOW) lower than 25 kg, it is only possible to fly at heights lower than 120 meters and always under visual conditions. Also, the flight has to take place in inhabited places or outskirts of the cities and with good meteorological conditions. These vehicles do not need to be registered at the "Registro de Matrícula de Aeronaves" or to have an airthworthiness certificate.

- For aircraft with MTOW beyond 25 kg, the applied regulations are the ones established in their Airworthiness Certificate issued by Agencia Estatal de Seguridad Aérea [6].

1.4 Goals

The main objectives of this project are presented below.

1. Understanding of all the quadcopter components and their effects into the vehicle dynamics.
2. Refinement of each component: calibration of the accelerometers, gyroscopes, speed controllers and radio control and PID tuning.
3. Selection of the best flight simulator for estimating theoretically the aircraft dynamics.
4. Commanding either manually or automatically a mission for the quadcopter.
5. Simulation of the missions through a flight simulator in order to be able to compare these results with the real ones.
6. Analysis and judgement of the results, with a detailed explanation of them and their causes, exploring the possibility of modifying the simulator equations for a better fitting to the real data.
7. Identification of the possible future work in order to continue advancing in the project.

1.5 Methodology

For accomplishing this project, firstly it has been defined a work methodology. It consists on establishing a guideline for ordering the different tasks which must be done at a specific time. It has been divided in three main time blocks.

- **Preliminary work**
  - Collection of quadcopters information, understanding the state of the art of UAVs and their applications evolution, together with their manufacture.
  - Studying the dynamical model for the quadcopter and its influence parameters.

  - Identification of the principal available software and hardware for the pretended study.

- **First work period**
  - First real flight tests with the aim to ensure the quadcopter ability to fly a mission and if possible, carry out the first manual missions.
  - Intensive analysis of the simulation environment, focusing in understanding the dynamic model implemented for the simulation.
  - Characterization of the quadcopter components for setting the simulator parameters.
  - Analysis of first results and check of work targets.

- **Second work period**
  - Manual flight tests.
  - Simulation of manual flight tests.
  - Improvement of the dynamic model of the simulation environment.
  - Autonomous flight tests.
  - Simulation with the default and the improved simulator of all the flown missions.
  - Analysis and understanding of the results, identifying future work for the project.

During this work planning, weekly meetings have been done for supervising the work. Moreover, according to UAVs legislation we were able to pilot manually the quadcopter during two hours in a reserved location at the university. Finally, it was defined a day for going to city ouskirts in order to be able to fly the autonomous missions.

1.6 Time planning

A Gantt diagram has been done in order to establish a time planning of the project. It is presented in Figure 1.2.

Assignment	Planned Start Date	Planned End Date	Real Start Date	Real End Date
Bibliography Recopilation	01/12/2015	29/01/2016	14/12/2015	26/02/2016
Quadcopter assembly and characterization	25/01/2016	19/02/2016	15/02/2016	11/03/2016
Simulation software selection and familarization	18/01/2016	26/02/2016	08/02/2016	11/03/2016
Software preparation for simulation	08/02/2016	25/03/2016	07/03/2016	08/04/2016
Manual experimental flight test	28/03/2016	20/05/2016	04/04/2016	28/05/2016
Autonomous experimental flight test	23/05/2016	31/05/2016	23/05/2016	03/06/2016
Simulation of flight tests	28/03/2016	31/05/2016	11/04/2016	03/06/2016
Results	02/05/2016	31/05/2016	01/06/2016	03/06/2016
Conclusions	23/05/2016	10/06/2016	23/05/2016	10/06/2016
Writting	18/04/2016	10/06/2016	02/05/2016	17/06/2016

Figure 1.2: Diagram of Gantt

1.7 Thesis structure

This thesis has been divided in several chapters. Its structure is presented below:

1. In the second chapter, it is analyzed the state of the art of UAVs. The different components possibilities for assembling a quadcopter and the different available simulators for predicting its performance have been identified. After considering the advantages and disadvantages of each of them, it has been selected the ones that are going to be used for this project.

2. The third chapter focus in the theoretical basis which is needed to carry out the tests. Firstly, it is analyzed the quadcopter physical model, in order to understand its flight capabilities. Then, it is presented the necessary steps for assembling the quadcopter used for the tests and how characterizing them for modeling it. Finally, the calibration steps needed for ensuring the correct flight of the vehicle have been explained.

3. In the fourth chapter, an analysis of the software used for the flight simulations has been done in depth. Firstly, the operation principle of the software has been explained. Secondly, a overview of the main libraries function used in the software has been presented, in order to understand better the program work operation. Lastly, the dynamic model used by the software has been analyzed.

4. Fifth chapter presents the project results. In first instance, it is explained the test procedure followed; and then, it is done a detaily analysis and discussion of the main results of the project.

5. Finally, some conclusions and future lines of research are drawn in the sixth chapter.

Chapter 2

State of the Art

The state of the arte of UAV has drastically changed in the last years. The number of projects where this kind of vehicles is involved has suddenly increased, aimed among other things by the rapid progress of electronic technology. The evolution in electronic devices such as sensors or data processing and integrating actuators have led to the production of miniature flying drones, which has extremely spread the potential applications of these vehicles.

Nowadays applications of UAVs are very wide. They are a great opportunity for research platforms, since they offer new performance possibilities due to their high manoeuvrability. Recent publications [7] [8] show that UAVs are being used for studying new control strategies based on non-linear systems and for creating and validating new matlab Simulink and CAD models.

Furthermore, it is also being researched how implementing new approaches for the creation and tracing of trajectories with obstacle detection and evasion. Developing a control system based in a vision sensor is another of the most important research fields of UAVs.

On the other hand, commercial aims and military and law enforcement applications have also been augmented. Commercial employment of UAVs goes from leisure activities to industry ones, as for instance using these vehicles for mapping and modelling. The relative low costs of using UAVs and their performance characteristics have been responsible of this trend. Also, social benefits are another factor, since for example if their mapping possibilities are used in mines, it results in a substantial increase in workers

security. Regarding military applications, UAVs are being used for rescue missions and surveillance purposes, either with old war applications.

As multicopters, especially quadcopters, are the UAVs of interest of this project, a deepest analysis of their actual state of the art is going to be done. It is going to be focused either in their physical building as in the most recent methods for simulating them.

2.1 Multicopters

Multicopters are aircraft lifted and propelled by multiple horizontal rotors. They belong to the rotary-wing aircraft.

2.1.1 Types of multicopters

Three main categories are found:

- *Quadcopter*: It has four horizontal rotors, two spinning clockwise and the other two spinning counter-clockwise. Its main characteristic is its high manoeuvrability.
- *Hexacopter*: It has six rotors. It is more stable than the quadcopter due to its greater number of rotors. Also, it is also able to flight with one motor failure.
- *Octacopter*: It has eight rotors. This is the most stable multicopter of the three analyzed categories, being able to carry on relatively large weights. In addition, it can fly with more than one motor inoperative.

For the aim of this project, it has been concluded that the features of a quadcopter are good enough for studying and simulating its dynamics, since the extra capabilities offered by the hexacopter and the octacopter do not have a great impact in the copter dynamics. In fact, the main conclusions can be valid for all these vehicles of the multicopter family.

2.1.2 Software

The software part of the multicopter consists basically on an autopilot based on an Inertial Measurement Unit (IMU). Two main choices are mainly been

used nowadays.

- *APM (ArduPilot Mega)*: It is a professional open source and customizable IMU autopilot based on the Arduino Mega platform [11]. It works at 100Hz frequency and it is capable of stabilizing automatically and navigating given the target waypoints. Moreover, it has a 4MB onboard data-logging memory. As a disadvantage, it does not incorporate an onboard compass.

- *3DR PIXHAWK*: It is a more advanced chip of the STM32 controller designed by PX4 open-hardware and manufactured by 3D robotics [12]. It has a port for microSD, letting to record high rate data-logging for long time. Its work frequency is 400Hz.

The 3DR PIXHAWK is faster and has greater memory capacity than the APM. It updates the sensor measurements every 400Hz while the APM does it at 100Hz and it can have as much memory as it is the microSD memory capacity. However, it is considered to be less reliable due to the recent use of this IMU in the multicopter investigation field. Therefore, due to the fact that the APM memory and speed available is able to withstand the operative requirements of this project, it has been selected the APM as the IMU autopilot of the quadcopter.

2.1.3 Hardware

An analysis of the most usual components of multicopters has been done, in order to select the ones to use in the quadcopter.

Main structure of the quadcopter

The function of the main structure is to be the framework of the quadcopter, gathering all the components. They are wanted to be as light as possible, but taking into account the need of support the stresses generated by the aerodynamic forces produced during the aircraft flight. The usual materials used are aluminium, fiberglass, some polymers and carbon fibre.

Motors

The most used motor for small multicopters is the Brushless DC motor. It works by the electromotive forces generated by the permanent magnet located at the rotor and the coil arrangement at the stators. Its main advantage respect to other possible motors, as the classic brushed DC motor, is that it do not produce magnetic interferences. This is particularly useful for improving the performance of the compass of the multicopter system. In addition, the higher efficiency of this kind motors has led to be the motor choice for the quadcopter.

Blades

Blades are the multicopter element in charge of producing the lifting and propelling forces. The main materials used for this component are plastics, carbon fibre and wood. Last one has been discarded due to the large rotation momentum generated by its high weight, which would notably affect the aircraft manoeuvrability. Concerning plastics and carbon fibre, both have similar features, although carbon fibre is a bit lighter. Finally, plastics blades have been selected, since they are cheaper and their effects in performance are negligible respect to the use of carbon fibre.

Electronic Speed Controllers

The aim of the Electronic Speed Controllers (ESC) is adapting the motor velocity according the PWM (Pulse Wide Modulation) inputs that they are receiving. For measuring the motor speed, it must be overseen the magnets positions. This is mainly done following two strategies: measuring the electromotive forces created by the magnets movement or using magnetic sensors based in the Hall Effect.

Battery

Multicopters commonly use LiPo battery as energy source. The main reasons of this tendency are their light weight, the large variety of shapes that they can adopt, their great capacities in comparison with their small size and their high discharging rates. Different capacities can be chosen for this type of batteries. Capacity is an indication about how much power can be stored by the battery, and it is measured in mAh (miliAmperes per hour). For

ensuring flight missions of 10 minutes, LiPo batteries of 3500mAh have been used.

Radio Controller

It is the device that connects the pilot with the aircraft and it is recommendable to be used also in automatic flight for being able to recover the multicopter control in case it is needed. It has a minimum of four channels and it transmits at frequencies in the range of 2.4 GHz. The chosen radio control has two sticks: the left one controls the throttle or vertical acceleration ($\ddot{z}$) and the yaw (ψ), and the right one controls the roll (ϕ) and the pitch (θ). The top right toggle stick is used for controlling the flight modes of the multicopter. The radio controller used can select a maximum of three modes. One of them is going to be reserved for AUTO mode. It will be selected when it will be being done an automatic mission. On this way, it can be recovered the quadcopter control as soon as it is required.

Telemetry and GPS

The function of the telemetry is proportionate flight data at real time. The GPS is in charged to estimate the global position of the aircraft by measuring the relative positions with respect to several satellites. It has to be remarked that the GPS selected for this project has to include a compass, since the IMU autopilot chosen does not include it.

2.2 Flight simulators

Simulators have been developed for analysing real situations without being exposed to any danger. Logically, it is preferable to suffer an accident when testing new machines in a simulated environment, where there are no consequences, than in real ones. One of the main features of this kind of software is that they are real time computing, since simulate results must be as close as possible to reality. Simulators development has increased considerably in the last years due to the dazzling evolution of computer technology, which has let simulators to be more complex and precise. Nowadays, simulators are being used in three main fields.

They are used for academia and research purposes. The simulation of

real machines or vehicles helps engineers to understanding their behaviour, so that improvements of the real machine or vehicle can be done through these simulated analyses. Moreover, simulators are currently being used for predicting theoretically object behaviours in environments where experiments cannot be done. For instance, before the Sojourner landed Mars, the real conditions of this mission, such as meteorological or gravity effects, were not known. So, simulations were extremely advantageous for estimating those conditions [13].

The second main field is the military and training one. The need to avoid danger situations for the beginners is the main cause for the use of simulators in this field. Lastly, they are being used for leisure activities. The computer games industry is the best example of this tendency.

In this project, a flight simulator is needed for emulate the physics of the quadcopter, together with the sensors operation and the earth environment. Three different alternatives have been studied:

- *Using free software.* These kinds of software are stored openly in Internet, where volunteers can develop it. Although it could be seem chaotic in a first insight, the great amount of testing users get these software to be improved rapidly, since many reports bugs are submitted. Among the free software possible choices, SITL (Software In The Loop) seems to be the best one, due to the fact that it has been developed primarly by APM for analysing APM firmware. Thus, it enables running Ardupilot code without needing any kind of software. One of its main features is the use of full sensor emulation.

- *Utilizing a commercial one.* These simulators have been developed and tested by individual companies.

- *Creating an own one.*

The commercial option was firstly discarded since they did not offer any special feature respect to the free software one. Then, the possibility of building an own simulator was studied. It was the choice that offered more flexibility when simulating. However, it implied a huge work load, considering that the APM environment has also to be simulated. Then, it was decided to use SITL for the quadcopter simulation, considering that it let simulating completely the quadcopter dynamics and the APM response and it gave the possibility to change their codes looking for simulation improvements.

Chapter 3

Quadcopter modeling, assembly and calibration

In this chapter, it is pretended to present the theoretical concepts that are behind the development of a quadcopter, as well as the components and the assembly process necessary for manufacture it. Then, the physical characterization of the quadcopter and its calibration process is going to be explained.

3.1 Quadcopter Model

The mathematical model for the quadcopter is going to be developed [14]. In the same way as any mathematical model, it is based on a series of assumptions:

1. The quadcopter is considered a rigid body consisting on a main airframe and four arms. Thus, the links between the arms and the airframe are assume to be ideal constraints, it means, there is no dissipation on them.
2. The arms meet at the center of the airframe at right angles.
3. The center of gravity of the quadcopter is located at the center of the airframe.

4. Blades forces can be modelled according to Momentum Theory [15].

These assumptions will be taken into account in the assembly process explained lately in this chapter, since it is required that the real quadcopter to be as similar as possible to the one described in this mathematical model.

3.1.1 Frames of reference

Three main frames of reference are needed to describe the motion of a quadcopter.

- The Earth reference frame. It is the one that is going to be considered inertial. Its origin is located at a static point of the Earth surface, being usually the home position of the aircraft. This reference frame will be notated by the subindex E: $[i_E, j_E, k_E]$.
- The body reference frame. It is set in the own quadcopter, and it has its origin in the center of mass of it. The x-axis and the y-axis of this frame are in the direction of the arms 1 and 2 respectively, according to the number criterion shown in Figure 3.1. The z-axis points the Earth in normal flight attitude conditions. The notation corresponding to this frame is: $[i_B, j_B, k_B]$.
- The blade reference frame. It is the one used for describing the propellers motion. Then, four of them must be utilized, each one at the propeller of each arm. Their origin is attached to their blades center of mass.

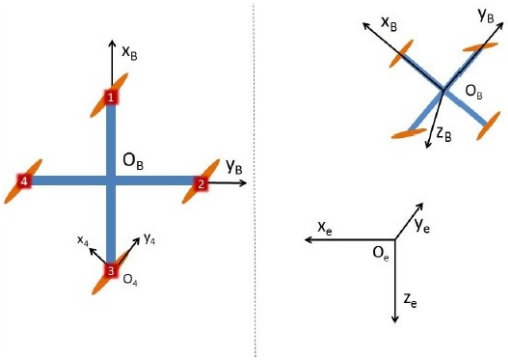

Figure 3.1: Reference frames used in the model deveopment of the quadcopter

Regarding the blade reference frame, two of them (arms 1 and 3 according to Figure 3.1) will be rotating clockwise while the other two will be spinning in the opposite direction in order to cancel the gyroscopic effects and the aerodynamic torques in normal flight attitude.

3.1.2 Mass geometry

The mass of the quadcopter can be described as the combination of the arms mass, the board and battery masses and the ones of the blades, due to the fact that they are the components with larger mass contribution. 0 subscript refers to the board and battery masses, which are located on the body axis origin; and b to the blades.

$$m_q = 4m_{arm} + m_0 + 4m_b \tag{3.1}$$

Considering the arms as infinitely thin bars and the blades as punctual masses located at the center of the blade reference frame, the inertial matrix of the quadcopter can be calculated. Moreover, due to the quadcopter symmetry, the principal axis of inertia will coincide with the body axis. Therefore, the diagonal moments of inertia of the quadcopter inertial tensor will be zero.

$$I = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} = \left(\frac{2}{3} m_{arm} l^2 + 2m_b l^2\right) \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{bmatrix} \tag{3.2}$$

3.1.3 Kinematics

For converting the vectors between the different reference frames, rotations matrices ae needed. The conversion between Earth reference frame and Body reference frame is particularly important when developing the equations of motion.

$$\begin{bmatrix} i_B \\ j_B \\ k_B \end{bmatrix} = R_E^B \begin{bmatrix} i_E \\ j_E \\ k_E \end{bmatrix} \tag{3.3}$$

$$R_E^B = \begin{bmatrix} cos\theta cos\psi & cos\theta sin\psi & -sin\theta \\ sin\phi sin\theta cos\psi - cos\phi sin\psi & sin\phi sin\theta sin\psi + cos\phi cos\psi & sin\phi cos\theta \\ cos\phi sin\theta cos\psi + sin\phi sin\psi & cos\phi sin\theta sin\psi - sin\phi cos\psi & cos\phi cos\theta \end{bmatrix} \tag{3.4}$$

Similar procedure must be carried out for relating the Euler angles with the body rotation rates, which are the ones measured by the gyroscopes.

$$\begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} = \begin{bmatrix} 1 & sin\phi tan\theta & cos\phi tan\theta \\ 0 & cos\phi & -sin\phi \\ 0 & sin\phi sec\theta & cos\phi sec\theta \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{3.5}$$

3.1.4 Forces models

Actions affecting the quadcopter include its own weight and the aerodynamic forces produced by its blades. These aerodynamic forces and moments must be modelled. With this aim, it is going to be applied Momentum Theory to the blades, arising the following equations:

$$M_j = \frac{1}{2} C_D \rho A \Omega^2 \tag{3.6}$$

$$T_j = C_T \rho \pi R^4 \Omega^2 \tag{3.7}$$

ρ refers to the air density, Ω is the angular velocity of the blade, R the radius of the blade, A the cross section of the blade, Tj the thrust and M_j the torque produced by the motors.

Once the thrust model is established, it can be model all the forces acting on the quadcopter [14].

Rolling Moments:

- Body gyro effect: $\dot{\theta}\dot{\psi}(I_{yy} - I_{zz})$
- Propeller gyro effect: $J_r \dot{\theta} \Omega_r$
- Roll actuators action: $l(-T_2 + T_4)$

Pitching Moments:

- Body gyro effect: $\dot{\phi}\dot{\psi}(I_{zz} - I_{xx})$

- Propeller gyro effect: $J_r\dot{\phi}\Omega_r$
- Pitch actuators action: $l(T_1 - T_3)$

Yawing Moments:

- Body gyro effect: $\dot{\phi}\dot{\theta}(I_{xx} - I_{yy})$
- Propeller gyro effect: $J_r\Omega_r$
- Roll actuators action: $(-1)^i \sum_{i=1}^{4} Q_i$

Forces Along x Axis:

- Actuator action: $(sin\psi sin\phi + cos\psi sin\theta cos\phi) \sum_{i=1}^{4}(T_i)$

Forces Along y Axis:

- Actuator action: $(-cos\psi sin\phi + sin\psi sin\theta cos\phi) \sum_{i=1}^{4}(T_i)$

Forces Along z Axis:

- Actuator action: $cos\psi cos\phi \sum_{i=1}^{4}(T_i)$
- Weight: mg

Q_i is the drag of blade i, l the arm length, J_r the blade moment of inertia and T_i the thrust of the rotor i.

3.1.5 Equations of motion

The equations of motion, according to Newton-Euler Formalism, are described with the following expression :

$$\begin{bmatrix} mJ & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} \dot{v}_B \\ \dot{\omega}_B \end{bmatrix} + \begin{bmatrix} \omega_B \wedge mv_B \\ \omega_B \wedge I\omega \end{bmatrix} = \begin{bmatrix} F_B \\ M_B \end{bmatrix} \tag{3.8}$$

Time derivatives must be done in the inertial reference frame, it means, the Earth one. Nonetheless, since vectors v and ω are projected in the body

reference frame, Coriollis Theorem must be used, which it has been taken into account in the second matrix of equation 3.8.

Finally, after a serie of algebraic operations that can be checked in [14], this set of equations of motion is achieved.

$$m\ddot{x} = (sin\psi sin\phi + cos\psi sin\theta cos\phi)\sum_{i=1}^{4} T_i \tag{3.9}$$

$$m\ddot{y} = (-cos\psi sin\phi + sin\psi sin\theta cos\phi)\sum_{i=1}^{4} T_i \tag{3.10}$$

$$m\ddot{z} = mg - cos\psi cos\phi \sum_{i=1}^{4}(T_i) \tag{3.11}$$

$$I_{xx}\ddot{\phi} = \dot{\theta}\dot{\psi}(I_{yy} - I_{zz}) + J_r\dot{\theta}\Omega_r + l(-T_2 + T_4) \tag{3.12}$$

$$I_{yy}\ddot{\theta} = \dot{\phi}\dot{\psi}(I_{zz} - I_{xx}) + J_r\dot{\phi}\Omega_r + l(T_1 - T_3) \tag{3.13}$$

$$I_{zz}\ddot{\psi} = \dot{\phi}\dot{\theta}(I_{xx} - I_{yy}) + J_r\Omega_r + (-1)^i\sum_{i=1}^{4} Q_i \tag{3.14}$$

3.2 Assembly

The assembly process of the used quadcopter is going to be presented. It is divided into two sections: the hardware assembling part and the wiring and connections part. Knowing the function and location of each component is useful to quickly localize errors in case of encountering a faulty piece.

The components that conform the quadcopter are the following ones:

- 4 x Quad Frame Arm
- Quad Upper plate
- 24 x M2.5*6 Mounting screws
- 16 x M3*8 Mounting screws
- 2 x Clockwise propellers
- 2 x Counter-clockwise propellers

- 4 x 6mm washer
- APM 2.5. Flight Controller
- 2 x Clockwise brushless motors
- 2 x Counter-clockwise brushless motors
- Radio Receiver
- GPS receiver
- Double-sided tape
- Cushioning sponge
- 2 x Neoprene Tape

3.2.1 Hardware assembling

ESCs and battery soldering

Firstly, the ESCs must be welt to the lower board. The red cables must be soldered in the positive (+) plugs and the black cables in the negative (-) ones. Secondly, the battery has to be connected to the board. With this aim, it has to be hooked two cables to the battery and then plug them in their respective positions. Again, a red cable and a black one must be used. A scheme of this installation is presented in Figure 3.2.

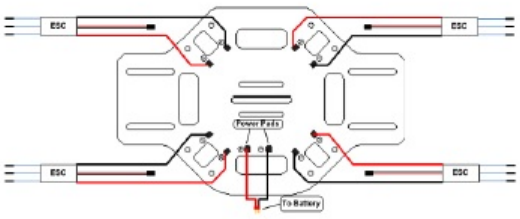

Figure 3.2: Scheme of the ESCs and battery soldering

Before proceeding with this installation, it is useful to check that all ESCs work properly. It can be done providing power to each of them and measuring

their voltage. This inspection is also advisable to be done at the end of the soldering, in order to verify that the soldering has been done properly.

Quadcopter frame assembly

The four arms must be attached now to the bottom frame. Four M2.5*6 Mounting screws must be screwed to the frame for each arm by using an Allen wrench. After that, the upper frame must be assembled on the other side of the arms. A picture of this final assembly is shown in the Figure below.

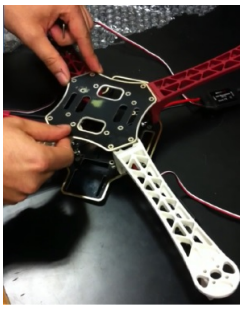

Figure 3.3: Graphical representation of the frame assembly

In this step, it is important to be care about the cables of the ESCs, ensuring that they can easily pass through the arm hole.

Motor assembly

The four motors must be assembled at the extreme of the arms using the M3*6 Mounting screws. When placing each motor, it has to be taken into account the motor rotation criterion which was established after developing theoretically the quadcopter model, due to the fact that the spinning of the vehicle around itself is wanted to be avoided. Thus, the motors with the same rotation direction (either clockwise or counter-clockwise) must be located in opposite arms, as it can be appreciated in Figure 3.4.

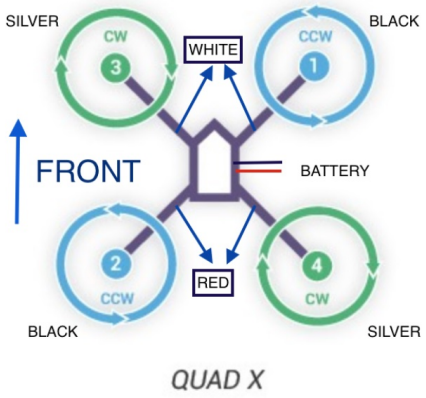

Figure 3.4: Motor numeration and location for the quadcopter

Once the motors are mounted, the blades must be installed. The same criteria than the one used for the motor applies to the blades: the ones which spin clockwise must be screwed on the motors that rotate clockwise and the other ones, which spin counter-clockwise, must be screwed with their respective motor pairs. In order to avoid any kind of confusion, it is presented in Figure 3.5 the shape of both kind of rotor blades.

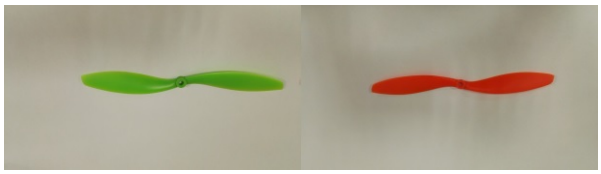

Figure 3.5: Counter-clockwise blade (left) vs Clockwise blade (right)

APM, radio controller and GPS assembly

Finally, the APM, the radio controller and the GPS must be gathered in the assembly. As vibrations affect seriously its performance, it is necessary to use some element for dampening these vibrations. Then, a cushioning sponge will be utilized with this aim for assembling the APM, neoprene tapes for the radio controller and a double-sided tape for the GPS.

Once the assembly is completely finished, it is recommended to check that the structure is rigid enough, so a revision to ensure that all screws are

correctly tight is suggested.

3.2.2 Wiring and connections

For the quadcopter to be ready for flight, it only remains the final step: proceeding with the wiring and the connections. It has to be connected the ESCs to the motor and the APM, and the GPS and the radio receiver to the APM.

Each motor has to be connected with its respective ECS. Again, it has to be taken into account if the motor is spinning clockwise or counter-clockwise, since the link with the ESC will be different depending on it. This difference is shown in Figure 3.6.

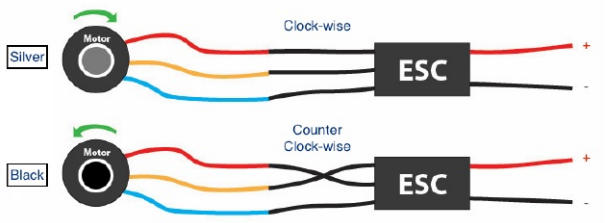

Figure 3.6: ESC connection depending on its spinning direction

Regarding the connection of the ESCs with the APM, the direction of rotation of each motor keeps playing a fundamental role. The numeration of the motors stablished before (Figure 3.4) shows what is the corresponding output pin for each motor.

Concerning the GPS connection, it must be followed two steps:

- Connect the GPS magnetometer port to the I^2C port of the APM.
- Connect the GPS port to the APM GPS port.

Lastly, the radio controller must be connected. On the one hand, the channels 1 to 6 of the receiver must be linked to the same numbers in the APM inputs pins. Please, note that they have to be connected in the raw indicated by a S. On the other hand, power must be supplied. Then, one wire must connect the positive pin of the receiver with the positive one of the APM; and another wire with their respective Ground pins.

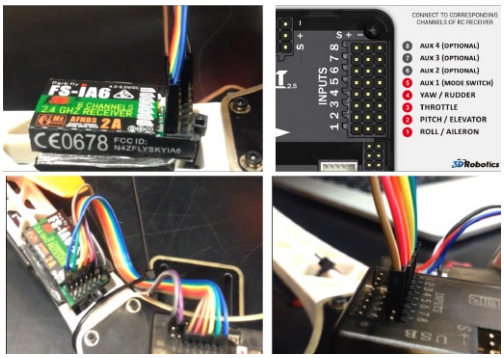

Figure 3.7: Radio receiver connection

It is important to fix all the cables to the body using zip ties, so as to avoid vibrations of the components. On this way, the quadcopter will behave closer to a rigid body.

3.3 Quadcopter properties characterization

It is necessary to adjust the parameters referring to the quadcopter, such as the blades moments and drag, or the mass and moments o inertia of the aircraft. A series of test bench are used for the experimental description of the blades performance. Concerning the aircraft moments of inertia, the commercial available CAD tool is found to provide enough accurate estimation.

Results for the quadcopter described are already available [18]. The procedure for obtaining them is going to be explained below.

3.3.1 Blades tests

The test bench made for the blade characterization consists on acquiring experimentally values of the thrust and the torque generated by the blades for different rotational velocities. Then, it can be linearized the results when comparing the torque (M_j) and the thrust (T_j) versus the rotational velocity to the power of two, obtaining the mean slope of those curves. The wanted coefficients are according equations 3.6 and 3.7 the result of dividing those

results between $\rho AR/2$ and $\rho\pi R^4$ respectively. The outcomes of this test bench are:

$$C_D = 3.41347 \cdot 10^5 \qquad C_T = 2.708042 \cdot 10^3$$

3.3.2 Mass and moments of Inertia characterization

For obtaining the mass, each individual component of the quadcopter has been weighted. The addition of all the weights let estimate the total weight.

Components	Quantity	Mass (g)	Total mass (g)
Upper frame plate	1	36.9	36.9
Lower frame plate	1	68.6	68,6
Arms	4	49.7	198.8
APM	1	33.1	33.1
Motors	4	53.2	212.8
ESCs	4	23.3	93.2
Receiver	1	8.1	8.1
Propellers	4	7.8	31.2
GPS	1	83.3	31.2
LiPo battery	1	191	191

Table 3.1: Mass of the components

Then, adding all the terms:

$$Final \;\; mass = \quad 0.957kg$$

In relation to the inertia moments calculation, it has been used a complete model of the quadcopter made in Solid Edge [18]. This program is capable of estimate the moment of inertia properties of the analyzed model when knowing the individual properties (its mass and its dimensions) of the components of the model.

	Moment of Inertia (kgm^2)
I_{xx}	$1.4201403 \cdot 10^{-2}$
I_{yy}	$1.3722055 \cdot 10^{-2}$
I_{zz}	$1.9468594 \cdot 10^{-2}$
$I_{xy} = I_{yx}$	$-4.3661 \cdot 10^{-5}$
$I_{xz} = I_{zx}$	$-3.4441 \cdot 10^{-5}$
$I_{yz} = I_{zy}$	$-1.79395 \cdot 10^{-4}$

Table 3.2: Moments of inertia of the quadcopter.

It is interesting to point out that the Solid Edge model of the quadcopter is not completely precise. Some geometry of the components has been simplified, as long as they have been considered of second order importance. Moreover, the wiring connections have been disregarded. The reasoning for this is the extremely lower mass of the cables respect to the rest of the quadcopter components, which makes their effect to be negligible.

Despite of this, results obtained seems to be coherent. Theoretically, I_{xx} and I_{yy} are expected to be equal, I_{zz} to be the double of I_{xx} and the rest of moments to be of secondary importance. It can be appreciated in Table 3.2 that these conditions are fulfilled.

3.4 Calibration

The calibration of the software of all the quadcopter components is needed before flying. Furthermore, the calibrated results parameters will be used also for the later simulation of the quadcopter flights. Mission Planner has been selected as the software for doing the calibration. It is going to be explained in detail the complete calibration process.

Firstly, the APM board must be connected to the computer via USB. Then, the APM must be linked to Mission Planner. After initializing Mission Planner, it is done by pressing the Connect button on the top right of the screen. For the APM to be recognized, it must be chosen before pressing the button the communication port. This can be done automatically by Mission Planner if the AUTO option is selected, or by clicking on the COM4 option and setting up the data rate to 115200 (Figure 3.8). It is important to remark that Disconnect button must be used before unplug the APM.

Once the APM is connected, it has to be gone to Initial Setup → Wizard

→ Mandatory Hardware. Then, the calibration process will start.

3.4.1 Frame type configuration

The frame of the quadcopter used must be selected. Mission Planner screen can be appreciated in Figure 3.8.

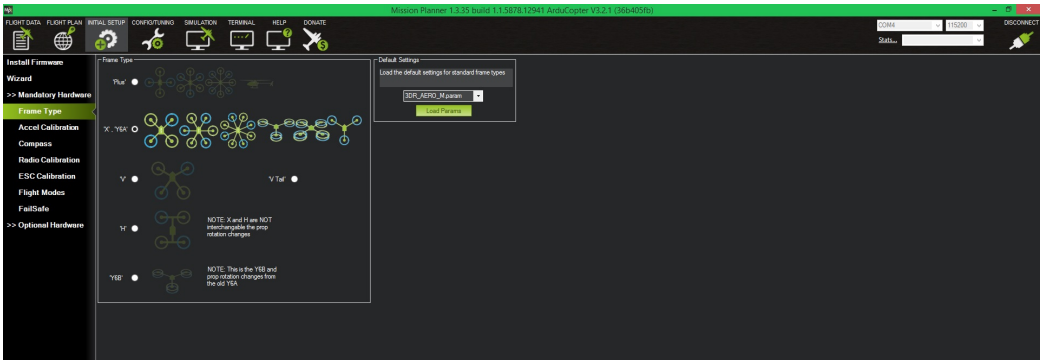

Figure 3.8: Mission Planner window display

3.4.2 Accelerometer calibration

Two steps must be followed:

1. *Calibrate Accel.* The autopilot will have to be placed on each edge: level, on right side, on left side, nose down, nose up and on its back.

2. *Calibrate level* It requires placing the autopilot horizontal.

After finishing these steps, the accelerometer offsets will be saved automatically. It is important to ensure that the autopilot is kept still just after pressing the key for each accelerometer calibration step.

3.4.3 Compass calibration

Press the Live Calibration button to start. The autopilot has to move around all axises in a circular motion. All the target white points must be touched in

this process (Figure 3.9). Once it is done, the calibration process will finish automatically, being displayed thee offset results. Typical offset values for the APM autopilot board are between -150 and 150.

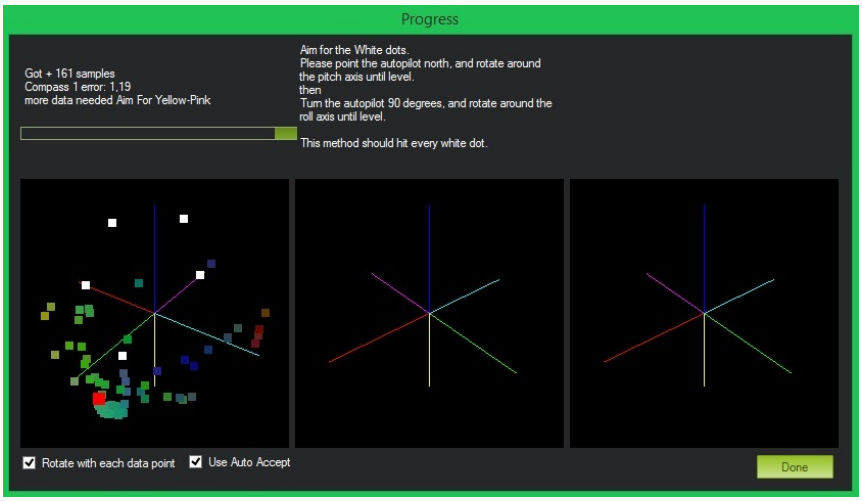

Figure 3.9: Compass calibration window

3.4.4 Radio Control calibration

The radio controller transmitter and receiver must be calibrated now. It is needed to teach the autopilot to work with it. Select Calibrate Radio button to start the calibration process. Then, both sticks must be moved in the largest possible circle in order to be attained their complete range of motion. The offset values will be pointed in the screen with a red line. Same procedure must be followed with channel 5 and 6 toggle sticks. Press end calibration to save the calibrated offset parameters. Note that if the screen bars are moving in the opposite direction to the radio control orders, it must be selected the reverse channel option on the radio transmitter.

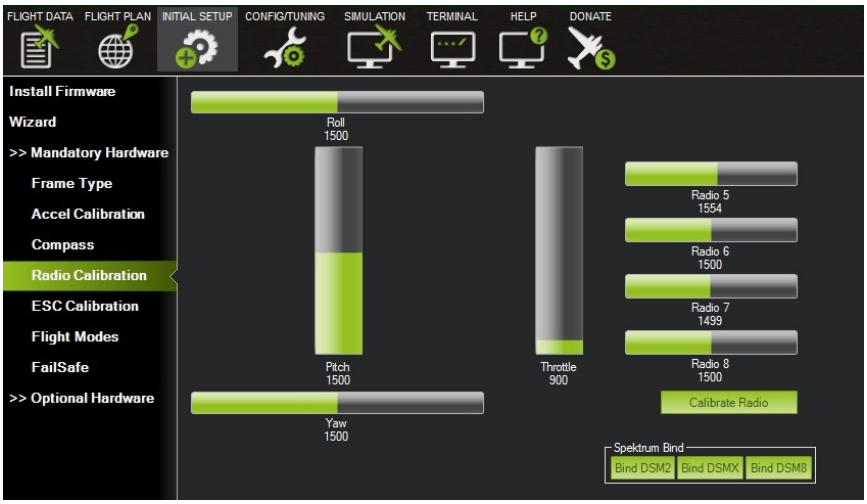

Figure 3.10: Radio Control calibration window

3.4.5 ESCs calibration

The minimum and maximum PWM values of the ESCs for maximum throttle radio controller commands. Before starting the ESCs calibration, the autopilot must be disconnected from the computer and the rotor blades must be disassembled. Then, it must be followed the next steps.

1. Turn on the Radio Controller and put the throttle stick at maximum.

2. Connect the LiPo battery. The autopilot's red, blue and yellow LEDs should light up in a cyclical pattern. If this condition is succeed, the APM will start in ESC calibration mode the next time the battery is plugged in.

3. Keeping the throttle stick high, disconnect and connect the reconnect the battery. The ESC calibration should start.

4. Wait for your ESCs to emit the musical tone. It depends on the battery's cell count, being that number the regular number of beeps that must be emitted (i.e. 3 for 3S, 4 for 4S). Then, additional two beeps to indicate that the maximum throttle has been captured.

5. Pull the throttle stick down to its minimum position.

6. The ESCs will emit a long tone indicating that the minimum throttle has been captured. Then, the calibration is complete. Check now that the motor spins by raising the throttle a bit and then lowering it again.

7. Set the throttle to minimum and disconnect the battery. The APM will exit the ESC calibration mode.

3.4.6 Flight Modes

Moving the toggle sticks of the radio controller it can be seen how the mode selected varies. Then, it must be selected the modes that are wanted for the missions. In the radio controller used in this project, up to three different modes can be saved. The ones chosen are the stabilize mode, for controlling manually the quadcopter through the radio controller; the guided mode, which enables the option to send commands to the quadcopter from Mission Planner; and the auto mode, for initializing automatic missions.

3.4.7 PID calibration

For calibrating the PID controllers, the mode Autotune will be used. It can be selected in the flight modes of the radio transmitter. Then, this mode must be activated once the vehicle is flying. The APM will start then to tuning the PID, lasting around five minutes. During this time, short corrections done manually using the radio controller sticks can be done for avoiding the aircraft to go too far.

Chapter 4

SITL (Software In The Loop)

A simulated environment is needed to make the test of the model created for the quadcopter. SITL software has been used with this aim.

4.1 SITL simulator

Software In The Loop (SITL) is a simulator that allows to run Ardupilot code without the necessity of any hardware. SITL was initially developed for Linux, but currently it can be also used in Windows. For another Operative Systems, as Mac OSX, it can be run on a virtual machine. The installation steps of this software can be found in [19].

The great advantage of this software is that it simulates the sensor data. It is obtained from the flight simulation of the flight dynamics models which SITL has in its libraries database. SITL has the ability to be run in several Ground Control Station programmes, such as MavProxy, Mission Planner or JBSim. In addition, it can be used to test a wide diversity of vehicles. The most important classes of them are [20]

- Multirotor aircraft
- Fixed wing aircraft
- Ground vehicles

The multirotor flight dynamics SITL models will be the ones used for the quadcopter test.

4.1.1 Ground Control Stations for SITL

For the interaction with this simulator, other software are needed, as MAVProxy, Flight Gear 3D or Mission Planner. Both MavProxy and Mission Planner are Ground Control Stations (GCS) for UAVs, basing their communication with them through the MAVlink protocol, which is the one used by the APM board. Flight Gear 3D is a graphical interface capable of displaying simulation information in 3D. There are more examples, such as JSBSim, but they are not relevant for this analysis since most of them can only simulate fixed wing aircraft. A description of their specific main features is provided for each software.

- *MavProxy* [21]: It starts running SITL for simulating both manual and automatic flight of the vehicle. For communicating with the user, this program uses command-line instructions. Then, the MaxProxy command prompt allows to modify the quadcopter parameters, change the flight modes of the vehicle, initialize a mission or send radio control inputs; the console, where information about the real time simulation is shown; and the 2D map, where it is graphically represented the trajectory of the quadcopter and its simulated position.

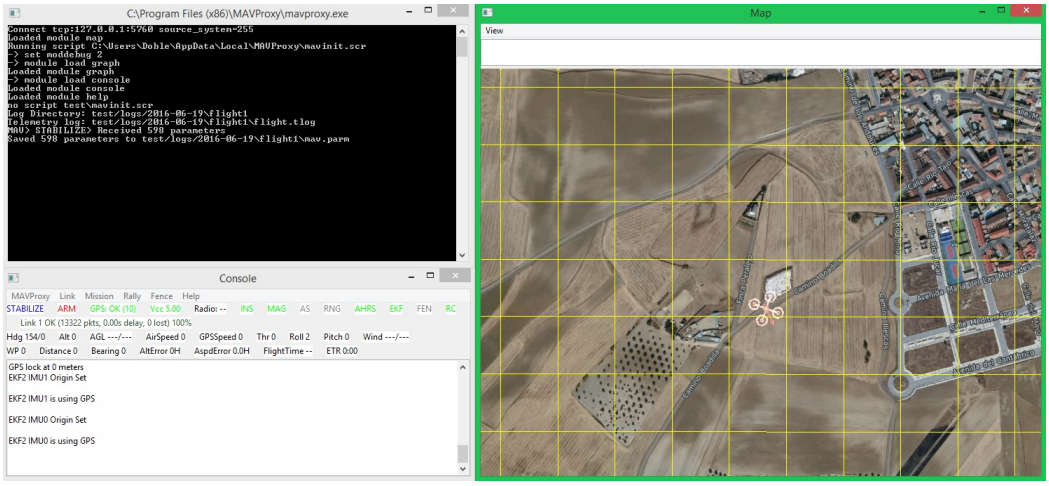

Figure 4.1: MavProxy graphical interface.

- *Mission Planner* [22] : It has similar features than MAVProxy. Both manual and automatic missions can be simulated. The main difference is that this software offers a more advanced Graphical User Interface (GUI), which is easier to learn by the user. It has an illustrated representation of the console, where it is shown graphically the attitude of the vehicle, together with velocity and position information. Moreover, Mission Planner can also be used for commanded mission, either manual as automatic, with real vehicles.

Figure 4.2: Mission Planner graphical interface.

- *Flight Gear 3D*: This software is characterized by providing a 3D representation of the simulated flight. Nevertheless, it must be pointed out that it has a limited amount of real environments which it is able to display graphically.

Figure 4.3: Flight Gear graphical interface.

The use of Flight Gear 3D has be discarded due to the fact that it was not capable to simulate in the location where the quadcopter tests were done. Regarding the simulation software chosen, it has been MavProxy, due to its capability of sending a set of recorded radio control inputs to the simulation. However, Mission Planner still will be used for other purposes, such as the quadcopter calibration or the log-data processing. Each of these actions will be indicated at each specific section.

Finally, it must be remark that these programmes can interact with other ones using different MavLink communication protocols, such as UDP (User Data Protocol) or TCP (Transmission Control Protocol). TCP is a more complex communication protocol, although it has the advantage to guarantee the data delivery. For the purpose of this project, both protocols can be used. The ports that must be utilized for these connections are:

- Port 14550 or 14551 for UDP link.
- Port 5760 for TCP connection.

A block diagram of the complete communication of all the programs is presented in Figure 4.4. It shows the function of every program in the quadcopter flight simulation. $\vec{x} = [\phi; \theta; \psi; x; y; z; \dot{\phi}; \dot{\theta}; \dot{\psi}; \dot{x}; \dot{y}; \dot{z}]$ is the state vector of the dynamics; Rc refers to the radio control inputs or outputs.

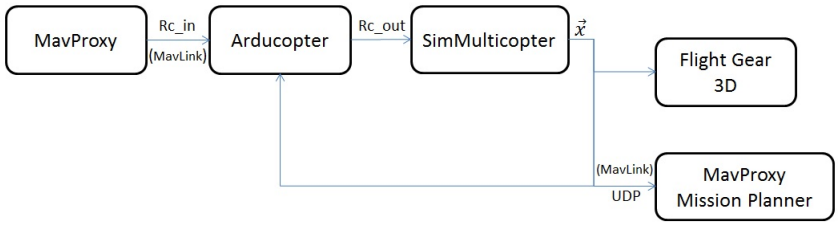

Figure 4.4: Block diagram of SITL performance and communication with MavProxy, Mission Planner and Flight Gear 3D.

4.1.2 SITL architecture

The general operation of SITL is going to be explained. It consists in an order of functions and libraries calls that it is repeated continuously in time.

Also, it varies depending on the flight nature, it means, if the vehicle is being controlled manually or automatically.

Manual flight

SITL operation cycle is repeated at 100Hz. In every cycle, the first thing happening is a check about the flight mode of the vehicle, updating it if it is necessary and calling that mode function. Then, that function starts running. This function interprets the user inputs, which in this case are received through the radio controller inputs, and converts them into a desired lean angles, rotation rate and climb rate. Those aimed angles and rates are sent then to the Attitude and/or Position control libraries, which are located in the AC_Attitude Control library.

AC_Attitude Control library can control the attitude position in several ways. The three most important ones are:

- Through *angle_ef_roll_pitch_rate_ef_yaw()* function. It works with roll, pitch and yaw angles referring to the Earth reference frame. Yaw angle corresponds with the vehicle rotation order, being positive when it must rotate to the right and negative when it has to do it to the left.
- Through *angle_ef_roll_pitch_yaw()* function. It also operates with angles in the Earth reference frame. The difference between last function is that the yaw angle indicates a vehicle rotation respect to the North direction. Positive values of the yaw imply East rotations while negative ones imply rotating West.
- Through *angle_ef_roll_pitch_yaw()* function. It works with the roll, pitch and yaw angles in the body reference frame.

All these functions follow the same command rule for the angles and rates values. The values used are the angle in degrees or the rate in degrees per second multiplied by a hundred. For instance, if the indication is that the aircraft must pitch up 10 degrees, the pitch angle value introduced in the function will be 1000.

Once it is called whichever of the aforementioned functions, the *AC_AttitudeControl::rate_controller_run()* is executed. It send the roll, pitch and yaw inputs to the AP_Motors library, using set_roll, set_pitch, set_yaw and set_throttle methods.

On the other hand, the AC_PosControl library is responsible of controlling the 3D position of the vehicle. It utilizes the *set_alt_target_from_climb_rate()*

method for stablishing the needed climb rate and altitude target and the *set_pos_target()* for declaring the offset distance from home. The climb rate is specified in centimetres per second and the home distance in centimetres. After that, the *update_z_controller()* function and/or the *update_xy_controller()*, depending on the axis involved in the operation, send the desired values to the AP_Motors library.

Finally, when the information processed by the AC_Attitude Control and the AC_PosControl libraries arrives to the AP_Motors, library, it is translated into absolute motor values, which are in PWM (Pulse Wide Modulation). PWM are modulation signal that encode the motor speed into a pulsing signal. These signals have two possible states: high (usually 5V) or low (usually 0V) voltage. The amount of time that the signal voltage is high every $2000\mu s$ indicates the value of the motor speed, being minimum for $1000\mu s$ of high voltage and maximum when the high voltage last the $2000\mu s$ time. Then, the Output_armed function is called, which checks the physical validity of the PWM obtained and passes these values to the AP_HAL libraries layer. These last libraries are responsible to send the PWM output to the right pin of the board. A schematic description of this operation procedure is presented in Figure 4.5.

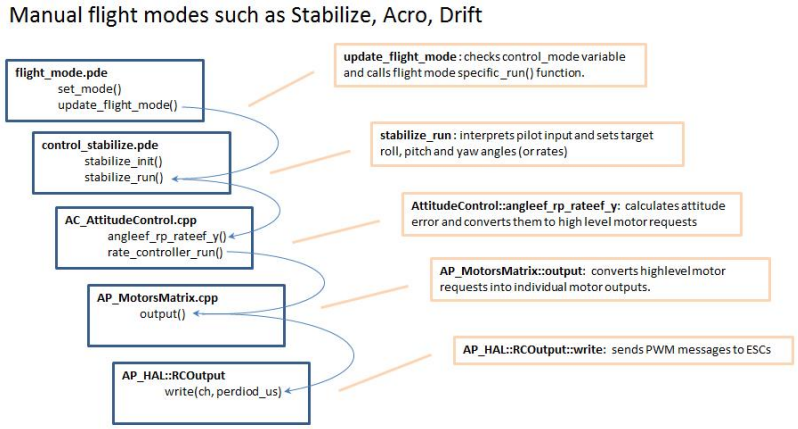

Figure 4.5: SITL operation for manual flight modes. [23]

Autopilot flight

Similar operation to the manual one is done. The cycle is also executed every 100Hz and it consists on checking and updating if necessary the flight

mode, interpreting the mission orders (in this case given by the autopilot) and translating them into desired angles and absolute velocities, processing them for being able to convert them into PWM values and lastly sending them to their respective board pins.

The difference with respect to the manual flight modes operation is found in the attainment of the desired angles, rates and absolute velocities. For automatic modes, the AC_WPNAW is used together with the AC_PosControl and AC_AttitudeControl ones. It is responsible of updating continuously the altitude, position and climb rate targets of the AC_PosControl library. For clarity, a scheme of this kind of SITL operation is shown in Figure 4.6.

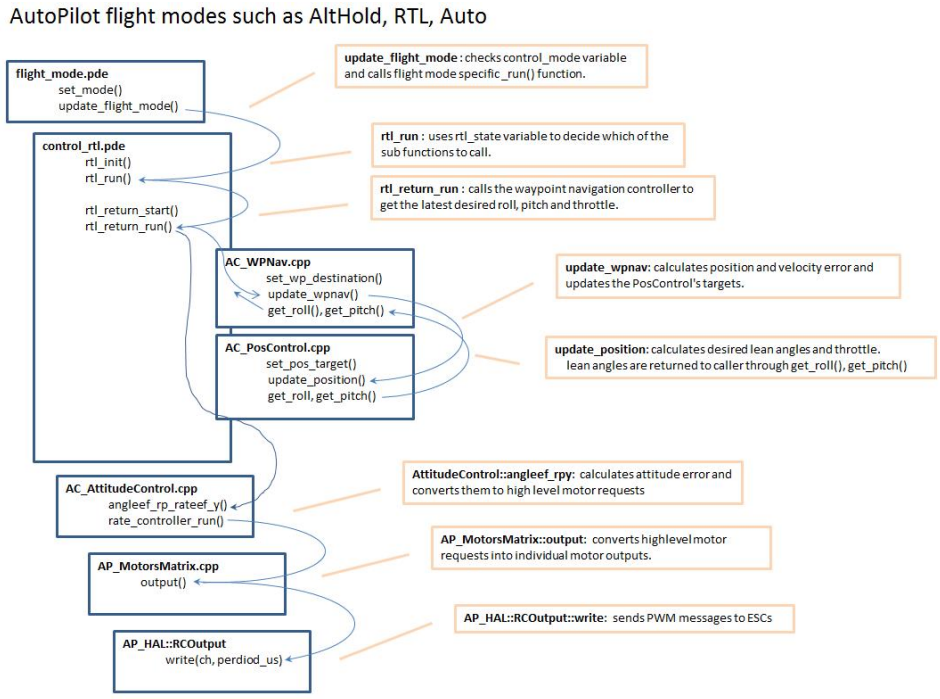

Figure 4.6: SITL operation for automatic flight modes. [23]

4.2 Arducopter Firmware Libraries

The general working behaviour of SITL has been explained. For being able to understanding this software in greater detail, it is interesting to analyse the SITL libraries. Specifically, it is going to be studied the main ones regarding

to the Arducopter SITL software, since that software is the one which is needed for analysing the quadcopter behaviour.

The SITL firmware is compounded by 630 archives, being 63 of them specific for Arducopter. The most relevant ones are going to be explained [24].

- *AC_AttitudeControl.* These libraries are in charge of defining the variables and functions necessary for the orientation and position control of the vehicle-
- *AC_Fence.* It defines the geometrical limits of the vehicle mission. Outside of these values, either due to the aircraft position or its altitude, the vehicle mode will change in order to come back to the home position. It is especially useful for using it as a safety backup.
- *AC_PID.* This library introduces the functions which define the controller algorithms. It includes the proportional (P) and the proportional, integral and derivative (PID) controllers.
- *AC_WPNav.* In this set of libraries is defined the variables and functions needed for the aircraft to be able to fly given some predetermined mission waypoints. It is used when the vehicle is flying in an automatic mode, being responsible of using the waypoints information for updating the target position of the AC_PosControl function (which is included in the AC_AttitudeControl libraries.)
- *AP_AHRS.* These libraries impliments the Attitude and Heading Reference System algorithms. It uses the gyroscope, accelerometer and magnetometer data for estimating the position and direction of the aircraft. The filter used in this calculation is the Direction Cosine Matrix (DCM) one. More advanced and complex filter, as the Extended Karman Filter (EKF), cannot be used due to the low calculation capacity of Ardupilot.
- *AP_Airspeed.* This library is responsible of the correct calibration and use of the speed sensors.
- *AP_Arming.* It realizes a check of the aircraft functionality before taking off. If this test is not succeeded, this function will not let the vehicle to arm.
- *AP_Baro.* These libraries implement the required functions for initializing the barometer, calibrating it and processing the data obtained by it.

- *AP_Common*. It includes the most usual functions needed by the rest of libraries, as for instance, the conversion from degrees to radians.

- *AP_Compass*. This set of libraries includes the necessary function for obtaining and processing the data given by the magnetometer models.

- *AP_Curve*. This library defines the relation between the PWM of the motors and its respective thrust generated

- *AP_GPS*. In this library it is implemented the different GPS models.

- *AP_HAL*. This set of libraries is responsible of relating most of the autopilot functions. It defines the pin inputs and outputs, the memory storage, the serial communication between the vehicle components.

- *AP_InertialNav*. This library has a function very similar to the AP_AHRS one. It calculates the altitude and the position of the aircraft according to the data obtained from the accelerometer, the barometer and the magnetometer.

- *AP_InertialSensor*. It controls the well performance of the accelerometer and the gyroscope of the APM board.

- *AP_Math*. The most relevant and used mathematical functions are implemented in this library, such as units conversions or matrices operations.

- *AP_Motors*. It includes the algorithms for defining the PWM inputs from the desired position, velocity, rates and angles.

- *AP_Mission*. This library interprets the mission scripts for the vehicle, where it is indicated the waypoints positions and the sequence to follow them. These mission scripts can be written using either Mission Planner or MavProxy.

- *AP_Navigation*. It contains information about how the aircraft must navigate through the waypoints, such as orientation angles respect those waypoints or the lateral acceleration that the aircraft must maintain while it is crossing them.

- *AP_Param*. In these libraries it is stored great amounts of data structures. This data correspond to the most usual variables used by the firmware.

- *AP_Rally*. It is responsible of creating the geometrical virtual points where the aircraft must go when the RTL mode is activated.
- *AP_RCMapper*. This library is used for converting the radio control inputs, given in PWM, to thrust and roll, pitch and yaw moments. It will be crucial for the manual simulated flights.
- *AP_Scheduler*. It controls the time available for each process. Also, it ensures that the operating loop is always working at the desired frequency, which in this case is 100Hz, as it was explained in SITL architecture section.
- *AP_SpdHgtControl*. The general functions shared by all the speed controllers are defined in this library.
- *AP_TECS*. The necessary functions for controlling both the altitude and the speed of the aircraft are located in this library. It is done using the total thrust for governing the total energy and the pitch angle for distributing this energy into kinetic and potential one, depending on what is required at each moment.
- *AP_Vehicle*. It provides the specific parameters for each kind of vehicles.
- *APM_Control*. This library actualizes the PID controller values used in the aircraft.
- *AP_OBC*. It activates the failsafe mode when it is needed.
- *Dataflash*. It enables operate with the data flash memory of the Arducopter firmware. It is particularly useful for saving data that cannot be read in real time.
- *Filter*. In this library it is defined the different filters that are used by the rest of libraries.
- *RC_Channel*. It manages both the input and output a data, which are given in PWM, that goes from the radio controller to the motors.
- *SITL*. It runs the Software In The Loop system, controlling the sensor data obtained by the vehicle simulation.
- *StorageManager*. This library is responsible of managing the memory storage of the vehicle board.

4.3 SITL quadcopter dynamic model

After analyzing the global operation and the main libraries used in the simulation process, it only remains to be studied the dynamic model used for the quadcopter. It can be found in the file *SIM_Multicopter.cpp*, located in the arducopter libraries, in the SITL folder (ardupilot/libraries/SITL). It is written in C++ language and works together with the *SIM_Multicopter.h* file, where all the functions and variables are defined. It is the dynamic model used for any kind of multicopter. Before doing the dynamics calculations, the multicopter type, which depends on its number of motors (quadcopter, hexacopter) and their location respect to the body reference frame (X or + configuration), is selected. The one corresponding to a quadcopter with X configuration will be the one used in the tests.

4.3.1 Model parameters

The dynamic model is based in four quadcopter parameters: its mass, its hover throttle, its terminal velocity and its rotational terminal velocity. In order to adapt these simulation parameters to the quadcopter used in the experimental tests, they are going to be adjusted.

- *Mass*: It is the total mass formed by all the quadcopter components. By default, a value of 1.5kg is used in the code, since most of today multicopters have masses between 1 and 2kg. The value that is going to be used is 0.997kg, since it is the mass of the tested quadcopter.

- *Hover throttle*: It is the weight to available thrust ratio of the quadcopter and it can have values between 0 and 1, being 0.51 the default one. Experimental tests have been done with the aircraft in order to estimate this parameter. These test consisted on flying missions where the aircraft was at hover most of the time. After that, the telemetry data was downloaded and the fraction thrust to total thrust was analyzed. Average of the results indicated that this parameter was around 0.61. Small modifications of that value have been done due for taking into account real effects. In fact, small imperfections of the rotor blades of the quadcopter and the different powers that the motors could receive from the different used batteries have led to lower thrust generations respect to the telemetry ones. Finally, after studying all these effects, the hover throttle values has

been selected to be 0.63.

$$\Pi_{hov} = \frac{mg}{T_{max}} \tag{4.1}$$

- *Terminal velocity*: . It is the maximum velocity that the aircraft can achieved and it used for estimating the drag contribution to the aircraft dynamics. It can be calculated by equalling the drag force to the weight of the quadcopter, which represents a free fall body motion. The value used has been 12.5 m/s, since it is the one corresponding to the phantom DJI, a similar quadcopter to the one tested. Due to the small velocities achieved during the tests, its influence it is expected to be minor.

$$D_t = \frac{1}{2}\rho C_A S V_t^2 = mg \Rightarrow V_t = \sqrt{\frac{2mg}{\rho C_A S}} \tag{4.2}$$

- *Rotational terminal velocity*: . It is the maximum rotational velocity that the aircraft can schieved. Again, it is used as an air resistance consideration for the aircraft motion. The value given by default has been used since all quadcopters have very similar values. It is $8 \cdot \pi = 25.13 rad/s$.

4.3.2 Equations of motion

The equations of motion of the quadcopter are written using the body axis reference frame in SITL. The conversion to Earth frame for the displacement calculations are done through the *SITL.cpp* script. Before entering in detail, it is necessary to define the variables used in the code.

- *motor_speed*. It is the thrust to total thrust ratio for each motor, so it can be interpreted as the thrust throttle of the quadcopter. It varies depending on the PWM values received by the formula presented in equation 4.3, having values between 0 and 1.

$$motor\_speed = \pi = \frac{T_{motor}}{T_{motor_{max}}} \tag{4.3}$$

- *thrust_scale*. It is the maximum thrust that can be extracted by each motor. Therefore, the thrust that each motor is producing can be deduced knowing the *motor_speed* and the *thrust_scale*.

$$thrust\_scale = \frac{mg}{N\Pi_{hov}} = \frac{T_{max}}{N} \tag{4.4}$$

- *motor_angle*. It is the angle between the motor of the quadcopter and the body axis. These values are defined when the quadcopter family type is selected in the simulation. With them, the moment that produced each motor can be calculated.
- *GRAVITY_MSS*. It is the gravity acceleration.

$$GRAVITY\_MSS = g = 9.80665 m/s^2 \tag{4.5}$$

The equations of motion are divided in two parts in the SITL code. Firstly, the accelerations due to the thrust forces and moments and secondly, the acceleration due to the air resistance or drag force.

Accelerations due to thrust force

These equations are into a loop from $i = 1 \ \ to \ \ number \ \ of \ \ motors$, which takes into account the different parameters of each motor. Then, the acceleration contribution is calculated for all the multicopter motors.

$$rot\_accel.x+ = -radians(5000.0) * sinf(radians(motors[i].angle)) * motor\_speed; \tag{4.6}$$

$$rot\_accel.y+ = radians(5000.0) * cosf(radians(motors[i].angle)) * motor\_speed; \tag{4.7}$$

$$rot\_accel.z+ = motors[i].yaw\_factor * motor\_speed * radians(400.0); \tag{4.8}$$

$$body\_accel = Vector3f(0, 0, -thrust/mass); \tag{4.9}$$

Equations 4.6-4.8 refers to equations 3.12-3.14, but they are simplified. The second order terms have been neglected, which implies that both the body and the propeller giro effect do not appear in the equations.

$$I_{xx}\ddot{\phi} = l(-T_2 + T_4) \tag{4.10}$$

$$I_{yy}\ddot{\theta} = l(T_1 - T_3) \tag{4.11}$$

$$I_{zz}\ddot{\psi} = (-1)^i \sum_{i=1}^{4} Q_i \tag{4.12}$$

rot_accel is the angular acceleration, expressing x, y and z the axis where the rotation is done, Therefore, it can be established that $rot\_accel.x = \phi$,

$rot\_accel.y = \theta$ and $rot\_accel.z = \psi$. $radians(5000)$ is a numerical value of the I_{xx} or I_{yy},l and the $thrust\_scale$. It seems to be assigned by given a typical value for multicopters of those parameters. Due to the fact that $I_{zz} > I_{xx} \simeq I_{yy}$, for $rot\_accel.z$ the numerical value is lower than for the other cases.

$$radians(5000) = 5000 \cdot \frac{\pi}{180} = 87.27 = 0(10) \simeq \frac{l \cdot thrust\_scale}{I_{xx}}$$

Using the values of I_{xx},l and the $thrust\_scale$ for the tested quadcopter, it has been checked that this conclusion seems to be right, since both values have similar order of maginitude.

$$\frac{l \cdot thrust\_scale}{I_{xx}} = 43.23 = O(10)$$

Then, $motor\_speed$ is the term for calculating the thrust with $thrust\_scale$, which is already embedded in $radians(5000)$; and the sine or cosine is the geometrical consideration needed for calculating the moment that the thrust force of each motor is producing, since the distance to the center of gravity is given by the arm length l by the sine or the cosine of the angle between the quadcopter body axis and the arm.

Lastly, the $motors[i].yaw\_factor$, it is the $(-1)^i$ of equation 4.12.

Regarding the body forces, the only one taken into account in this model is the effect of the thrust. It only actuates on the z-axis due to the fact that it is assumed that the force produced by the rotating blades is perfectly perpendicular to the rotation direction, which is occurring in the body x-y plane. The $thrust$ term of equation 4.9 is the resultant of summing all the motor forces.

$$thrust+ = motor\_speed * thrust\_scale; //newtons$$

Air resistance or drag force effect

A linear model has been established in SITL for calculating the accelerations due to drag. For taking into account the drag due to the body rotation:

$$rot\_accel.x- = gyro.x * radians(400.0)/terminal\_rotation\_rate; \quad (4.13)$$

$$rot\_accel.y- = gyro.y * radians(400.0)/terminal\_rotation\_rate; \quad (4.14)$$

$$rot\_accel.z- = gyro.z * radians(400.0)/terminal\_rotation\_rate; \quad (4.15)$$

In order to look for the equations responsible of these expressions, it is calculated using Linear Aerodynamics to estimate the drag.

$$D = \frac{1}{2}\rho C_A S V^2 \quad (4.16)$$

The velocity during the rotation is the angular velocity times the radial distance to the center of rotation. For calculating the moment produced by the drag when the body is rotating, a characteristic length x_c is introduced.

$$\ddot{\phi}_D I_{xx} = D x_c \quad (4.17)$$

$$D_t \cdot x_c = Tl = \frac{1}{2}\rho C_A S V_t^2 = \frac{1}{2}\rho C_A S \omega_t^2 R^2 \quad (4.18)$$

Combining equations 4.16 and 4.18, the following result arise.

$$D = \frac{Tl}{x_c}\left(\frac{\omega}{\omega_t}\right)^2 \quad (4.19)$$

Lastly, operating with equations 4.17 and 4.19, the angular acceleration due to the drag is obtained.

$$\ddot{\phi}_D = \frac{Tl}{I_{xx}}\left(\frac{\omega}{\omega_t}\right)^2 \quad (4.20)$$

Similar analysis can be done for the other angular accelerations, being the moment inertia term the only difference between them.

$radians(400)$ is a numerical value for the part of the equation different from the angular velocity ratio. The difference respect to the theoretical model is the fact that the rotational velocities are not elevated to two.

For the translational drag, the following expressions are used. As the accelerations are calculated in the body reference frame, firstly it is obtained the velocity respect to the Earth reference frame, since drag depends on the absolute velocity of the quadcopter and then, it is converte to the body reference frame through $aircraft.get\_dcm().transposed()$ function. This function describes the rotational matrix of equation 3.4.

$$Vector3f air\_resistance = -aircraft.get\_velocity\_ef() * \\ *(GRAVITY\_MSS/terminal\_velocity); \tag{4.21}$$

$$body\_accel+ = aircraft.get\_dcm().transposed() * air\_resistance; \tag{4.22}$$

Linear Aerodynamics is used again to calculate the drag. Combining equation 4.16 with equation 4.2, the one of the terminal velocity, it can be deduced the drag coefficient.

$$C_A = \frac{2mg}{\rho S V_t^2} \tag{4.23}$$

Then, the accelerTion due to the drag force can be calculated introducing this result in equation 4.16.

$$D = mg \left(\frac{V}{V_t}\right)^2 \Rightarrow a_{Drag} = g \left(\frac{V}{V_t}\right)^2 \tag{4.24}$$

Again, the equations coincided, except the velocity relation, which should be multiplied to the power of two. Therefore, these models are not completely accurate. However, these calculations are conservative, since the real accelerations should be lower than the ones calculated, since the velocity ratio will be usually a number between 0 and 1 so the square of this ratio will be lower. Despite of this, the errors due to this factor are expected to be low, because of the low velocities of the quadcopter during the tests, which would make these accelerations to be negligible.

Chapter 5

Results

This chapter is going to be divided in two sections. At first, the procedure that has been followed for doing the experimental flight tests and the simulations is going to be explained . After that, it is going to be presented the results of those tests. The results are going to be compared according the dynamic equation of motion used for the flight simulation.

5.1 Test procedure

The objective of the tests that are going to be presented is using an existing flight simulator running the SITL arducopter code for simulating quadcopter flights. These flights are also going to be flown by a real quadcopter. Then, the telemetry data of both cases are going to be compared. On this way, it is going to be analyzed if the simulation results are suitable for testing the performances of the quadcopter. Finally, the equations of motion of the simulator software are going to be modified in order to improve the simulation capability of the arducopter software.

Two kinds of tests are going to be done: manual tests, where the quadcopter receives the commands through the radio controller; and automatic tests, where the quadcopter is auto-guided to follow a predefined mission.

5.1.1 Manual tests

Several missions are going to be defined to fly manually. Each of them will have a specific objective, such as making a 360° yaw or pitching up, in order to test lately the simulator capability for predicting these movements. The steps that are going to be followed are explained below.

Firstly, the manual mission is going to be done with the real quadcopter. The commands will be sent by the pilot through the radio controller. Then, the data-logs of the mission have to be extracted from the APM board. Mission Planner is going to be used for this task. Once the APM is connected via USB to Mission Planner, the data flash logs stored in its memory can be download as follows.

1. Go to "Flight Data→DataFlash logs→Download DataFlash Log Via Mavlink".
2. A window will be opened with all the data logs saved in the APM memory. Select the save option.
3. Go to Mission Planner "Create matlab file" option in Flight data Data Flash logs. Then, the downloaded *.log* data files are going to be converted into *.mat* files, which will let to work with them using Matlab.
4. The name of these logs is going to be changed to MissionX, where X is the number which identifies the mission, in order to be able to work easily with this data later.

It must be pointed out that the initial data values of the data logs files are unrealistic, due to the fact that the initialization and calibration of the measurement components do not take place until the quadcopter is armed. A filter has been built for eliminating this data, which can be seen in Appendix A. The point at which the quadcopter was armed can be easily identified, since the height measured by the barometer initializes to zero. Then, the time at which this occurs is going to be the reference time for making the data filter. The main variables are going to be saved in a file called MX, being X the mission number.

With the filtered data file, the PWM radio control inputs have to be extracted. They are saved in a variable called RCIN. A *.txt* file called *MissionX.txt* (where X means the mission number) has to be created for sending the commands later to the simulator. The script code that does it is in Appendix B, and it is based in the one developed by [25]. Moreover, it

is necessary to send those PWM radio control inputs at the time they were sent in reality. It is done by using the *tdelayCreator.m* function (Appendix C), which creates a vector called *TimeVectorX* (again, the X represents the mission number) with the time difference between the radio control input data. Then, either the *MissionX.txt* and *TimeVectorX.mat* files are called through a python function which is responsible to interact with MavProxy simulation.

For simuating the flight, firstly the simulator has to be started. SITL flight simulator is called from the Cygwin terminal. At the same time, MavProxy GCS is connected to the simulation in order to able to send commands to the vehicle. The following commands are written in that terminal.

$$cd \sim /ardupilot/ArduCopter$$

$$sim\_vehicle.sh - j4 - fX - LUC3M - - map - - aircrafttest - - quadcopter$$

The first one is necessary for indicating that the vehicle simulation must use the dynamics written in the *SIM_Multicopter.cpp* script. With the second one, it is indicated the location of the test and the type of vehicle used, which in this case is the quadcopter. It is necessary to remark that it also has to be pointed that the frame disposition of the tested vehicle is the X, due to the body axis reference frame position respect to the quadcopter arms. This is done by the command "$-fX$" introduced in the Cygwin terminal command presented before.

The location indicates the point where the flight is going to be started. New location places can be used by written its coordinates in the *location.txt* file located in ardupilot/Tools/autotest folder. The first two number corresponds to the latitude and longitude position, the third number to the ground elevation from sea level and the fourth one to the starting quadcopter orientation, being 0 the North direction.

$$UC3M = 40.3332942735714, -3.765718942876447, 666.59, 0$$

After MavProxy is initialized, the parameter setting coming from the quadcopter calibration has to be set. It must be written in the MavProxy command prompt the following command with the name of the file of the quadcopter parameters.

$$param\ load\ ../Tools/autotest/MyCopter.param$$

Then, the python function *TimeVecudp.py* (Appendix D) is called, and it interacts with Mavproxy by establishing an UDP connection. After that, the vehicle is armed and then, it calls the PWM input and time vector scripts for sending the PWM values to MavProxy. Once the simulation is finished, the flight simulator must be closed. The simulation log is recorded automatically. This log is saved in *ardupilot/ArduCopter/logs*. Then, it has to be converted as before to *.mat* in order to be able to work with it.

During the simulation, there are some small delays due to computer processing of the information. Due to this fact, the simulated mission time does not fit the real one exactly. Therefore, in order to understanding the simulation results it has been assumed that the time for the data log is equal to the time which last the radio control input commands. This time data processing, together with the representation of the results, is done by *GlobalComparisonManual.m* script, which can be found in Appendix E. Lastly, it must be taken into account that this script uses the real mission initial conditions for yaw for establishing the initial yaw angle of the simulation. The yaw response of the vehicle is not influenced by the initial yaw condition, so the simulated results for the yaw motion variation do not change. Then, setting the initial conditions simply helps the results comparison.

5.1.2 Automatic tests

Similar procedure is going to be followed for the automatic flight. Several missions are going to be defined, although this time they are going to be named as MissionXAuto, where again X is the number of the mission. Firstly, the flight with the quadcopter is going to be done and lately, the mission is going to be simulated.

For flying the automatic mission, the target mission has to be recorded in the APM board of the quadcopter. The mission can be written using Mission Planner, in the Flight Plan option. Then, the APM board must be connected to Mission Planner and pressing the "Escribir WPs" button, the mission is recorded on the APM board. For safety reasons, the automatic mode is going to be selected through the radio control, in order to be able to recover the vehicle control at any moment if needed. In normal conditions, only two steps are needed to start the mission using the radio controller. Firstly, the quadcopter must be armed and after that the Auto mode must be selected through the radio control. The mission will start taking off, the quadcopter will follow the target waypoints and then it will come to the initial position

to land. Lastly, the data-logs must be downloaded from the APM board and converted to .mat files using Mission Planner.

The next step is simulating the mission. Similar steps than the one needed for the manual flight have to be done. SITL flight simulator is going to be initialized through Cygwin terminal, together with MavProxy as GCS. This time, a different location is going to be selected, since the automatic missions have been done in a different place. The new command to be introduced is:

$$sim\_vehicle.sh - j4 - fX - LCRRU - -map - -aircrafttest - -quadcopter$$

This location has been also added to the *location.txt* file, since it was not there by default.

$$CRRU = 40.167702, -3.836535, 625.400024, 161$$

The quadcopter simulation is going to be initialized then in the same starting position than the real one. The heading angle 161^0 has also been selected for the initial yaw angle of the quadcopter to coincide with the one of the real missions.

MavProxy console will have started. Again, the calibrated parameters of the quadcopter have to be introduced. This time, it is also necessary to load the automatic mission. It is done by typing the following command.

$$wp\ load\ ../Tools/autotest/MissionName.waypoints$$

Finally, the quadcopter must be armed and guided mode must be used for taking off. This must be done for avoiding the simulator to disarm the vehicle. Therefore, it is going to be a small take off of 0.5m and immediately after the aircraft start to fly, the auto mode is activated. Then, the aircraft will finish the taking off specified in the mission and will complete all the mission requirements, which includes flying through the waypoints and coming back to the initial position to land. For treating the resultant logs, *GlobalComparisonAuto.m* script has been written. It uses the logs after converting them into *.mat* files, which is done by utilizing Mission Planner as before. This script can be found in appendix F.

5.1.3 SITL equations of motion

The simulations have been done for two different set of equations of motions: the ones used in SITL by default and new ones. The default equations of

motion which are implemented in SITL have been explained in chapter 4.3. Due to the fact that those equations are a simplification of the quadcopter dynamics, they have been modified.

The generic values appearing in the default equations have been replaced by the ones corresponding to the tested quadcopter. Moreover, the gyroscope effects have been added and the drag calculations have been improved. In fact, the quadratic relation between the velocity and the drag force has been taken into account, opposite to the linear model used by default. Equations 3.9-3.14 have been added to the *Multicopter.cpp* script of SITL. The complete modified script can be found in Appendix G.

Finally, it must be poined out that both simulations use some modified parameters, in order to adjust the dynamics to the tested quadcopter. These are the vehicle parameters that SITL accepts for the simulation. It has been explained in detail in chapter 4.3. However, as a reminder, those values are going to be presented in table 5.1.

Mass (kg)	Hover Throttle (-)	Terminal Velocity (m/s)	Rotational terminal velocity (rad/s)
0.997	0.634145	12.5	8π

Table 5.1: Quadcopter parameters for the SITL dynamic equations

Firstly, the result for the manual flight missions will be presented and then, those for the automatic ones.

5.2 Results

In order to understand in a better way the influence of modifying the equations of motion, it has been compared the results obtained by both simulations.

5.2.1 Manual flights

Three missions have been done for testing the simulation capabilities. In each of them, there have been different flight objectives, in order to be able to test all the simulator possibilities. It has been chose the quadcopter angles (roll, pitch and yaw) and the altitude as the reference variables, since they also

contain the information of the speed and the acceleration suffered due to the fact that they are obtained by integrating the acceleration measurements; which are the ones measured by the sensors in the real flight and calculated by the dynamic model in the simulated flight.

Mission 1

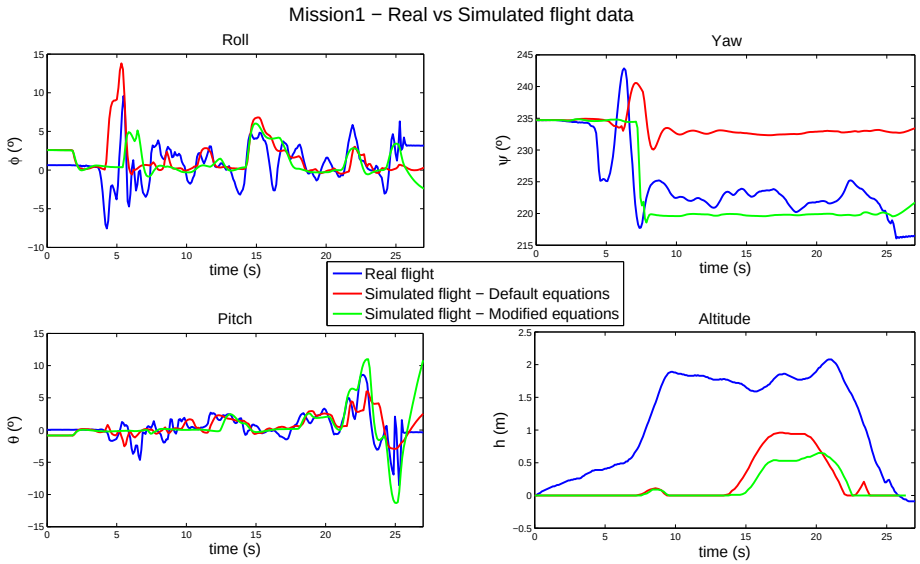

Figure 5.1: Mission 1 - Real vs Simulated flights data comparison.

It can be observed in Figure 5.1 how similar behaviour to the one of the real flight is obtained in the simulation. Worse flight predictions are done by the simulator when SITL default equations were used. For this simulation case, roll angles peaks are quite coincident, so the main turns of the quadcopter are well catched by the simulator. Regarding the yaw angles, it can be appreciated that there is a small variation in the final angles. It seems to be an indication that the moment of inertia about the z-axis, the one which affects predominantly in the yaw motion is not well estimated in the default equations.

Looking at the pitch variation, it can be seen that the general trends of this angle during the flight is captured, although the main peaks are not well predicted. This fact can be a potential hazard, since it is needed to

accurately predict the maximum pitch angle of the quadcopter in order to be able to use the simulator to find if a determined manoeuvre is possible. Lastly, it can be seen that the altitude error is large. It seems to be due to the fact that GPS measurements are being used for estimating the altitude, which usually has errors of 1 metre magnitude. So, due to the fact that very small altitudes are being attained during the tests, this error seems to be expected.

When focusing on the simulation using modified equations results, it can be appreciated how the body orientation in space, defined by the roll, the yaw and the pitch angles, is very similar to the one in the real flight. The mean peaks in roll and pitch coincide, so the simulated results are conservative in this case, since slightly higher peaks are predicted respect to the real ones. With reference to the yaw, it can be seen how the final reached angle is obtained, being eliminated the 20° error occuring in simulations with the default equations. Looking at the altitude, similar errors to the one obtained before are found. It seems that it is also caused by the very low altitudes that are reached during the test.

Therefore, orientation prediction is improved when adapting the equations of motion to the tested quadcopter. The roll and pitch peaks, together with the simulated yaw angle are closer to the real results for the modified equations. Regarding the altitude, similar errors are found. Nevertheless, it can be seen that the simulation using the modified equations predicts in a better way the vertical movement of the vehicle, since its altitude increments when it does the real case (at 16s from the take-off).

Mission 2

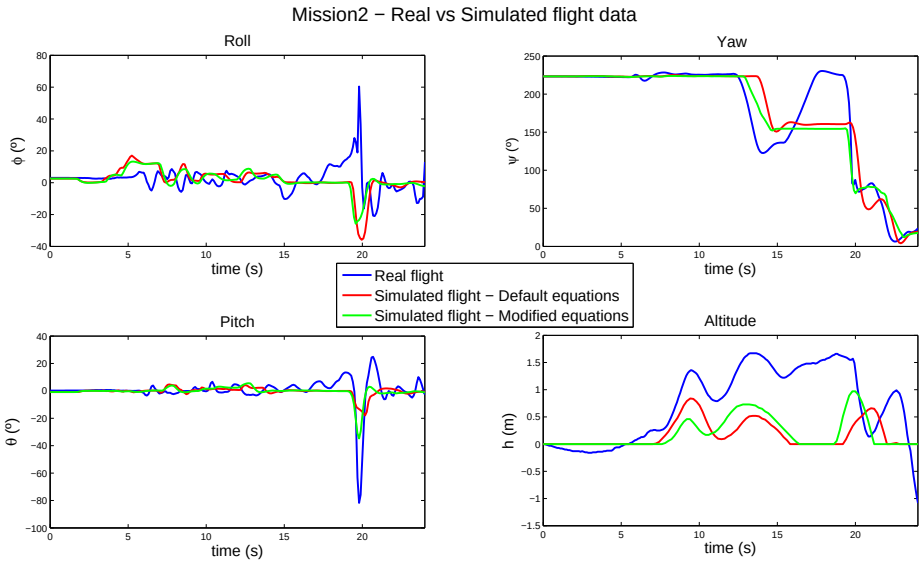

Figure 5.2: Mission 2 - Real vs Simulated flights data comparison.

In Figure 5.2 can be appreciated the test results. Simulator results are smoother that the real ones. It seems to be due to the absence of external disturbances such as wind or gusts in the simulated environment. The great negative roll at time equal to 20s in reality is similar to the one obtained in the simulations, specially for the simulation using modified equations of motion. However, the huge positive peak is not got. The large value reached in the real flight seems to be a sensor measurement error, since that measurement is too sharp. Then, a rapid acceleration of the quadcopter might be the cause for the sensor to interpret such high roll value. Looking at the yaw angle results, it can be osbserved how the two yaw step motions made in the real flight are simulated by both simulations. Nevertheless, the final simulated yaws are different to the real ones in the case of the default equations simulation. Again, a bad estimation of the quadcopter moments of inertia in this model case seems to be the cause of this error, since the final angles after each movement very close to the real one for the model using the modified equations. Concerning the pitch angles, it can be seen how the great nose down of the quadcopter is well predicted, although such large value is not reached in the simulation. Sensor error due to large acceleration

in small time period seems to be the main cause of this difference. In fact, quadcopters do not descend at pitch values near to 80^0. It must be pointed out that this pitch minimum value is better predicted by the simulation using modified equations, since it predicts that peak at the same instant of time than the real flight.Lastly, with regard to the altitude variation, similar errors to the one encountered in Mission 1 are found. Nonetheless, the flight tendency (ascending or descending) is well simulated when it is compared with the real result.

Similar conclusions to the ones reached when analyzing Mission 1 are obtained when comparing the results for both simulations. The roll, pitch and yaw angles are better predicted by the simulation using the modified equations of motion, since its results are closer to the real ones. In fact, this analysis can be justified with the peak roll angles at 20s of the mission, the better prediction of the negative pitch suffered and the closer values of the transition and final yaw angles. Referring to the altitude, similar absolute errors are found again. However, better vertical position is predicted by the simulation using modified equations of motion. When higher altitude positions were reached during the real flight, the simulation using modified equations of motion also attains the larger altitude values. Moreover, the ascending and descending tendency is better captured by this model.

Mission 3

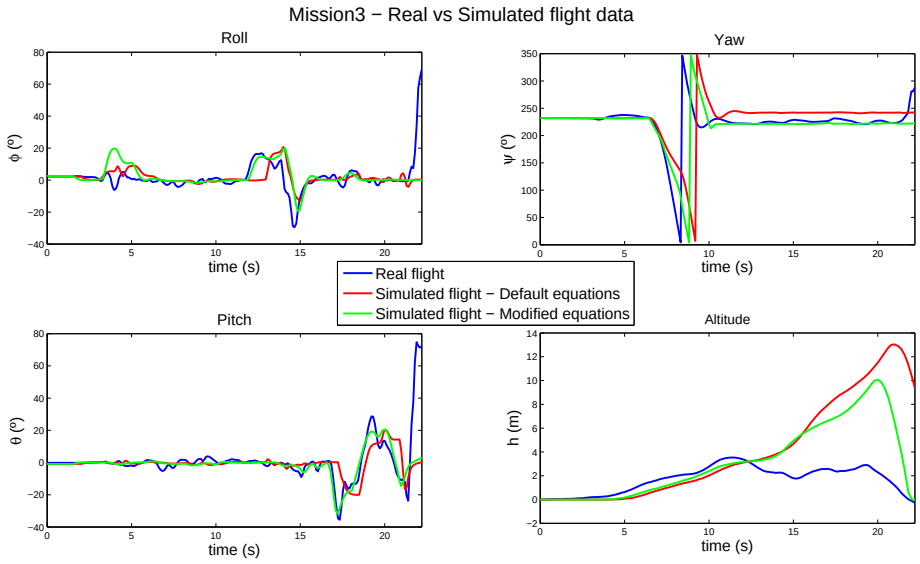

Figure 5.3: Mission 3 - Real vs Simulated flights data comparison.

Similar results to the ones obtained in the missions before are reached in Mission 3. They can be observed in Figure 5.3. Both main roll and pitch variations are simulated, although larger errors are found in the default equations simulation. The mean peak values of the test are coincident in time, and their absolute value are very similar. Referring to the yaw, the direction of the motion to the left is well simulated for both models. The 360° yaw made in the test has been obtained in the modified equations simulation, although a small delay appears in the simulated motion. It seems to be caused due to a lower air resistence than the one predicted during the real flight. The altitude error is quite big. In fact, the quadcopter has not landed in the default equations simulation when the real flight has finished. The low altitudes reached during the test might cause this error. However, slightly better results are simulated whe using the modified equations. The final descent commence at the same time (around 19.5s from the take-off) and that the landing time is similar for this case.

Therefore, it can be concluded that similar results are found when the modified equations are used for the simulations. Roll and pitch both negative and positive peaks are better predicted by the simulation with modified

equations, which it is coherent with what was expected, since the modified equations takes into account in a deeper way the quadcopter dynamics. Regarding the yaw, it can be observed how the simulation is perfect for the modified equations case, having also smaller delay. Finally, looking at the altitude, it can be seen that the simulated case with modified equations have lower absolute error. Moreover, for this case the land time is much more closer to the real result.

5.2.2 Automatic flights

Several missions were defined for the automatic flight tests. These test missions consisted in automatically taking off, following some waypoints varing the target altitudes and returning to the take-off position for landing. Due to the similar results obtained, the results for the first mission is presented, due to the fact that same conclusions are reached in the other missions. Since the mission consists on fliying through some waypoints, the latitude and longitude position of the quadcopter becomes of primary importancy. Then, the pitch, roll, yaw and altitude results are presented together with the quadcopter displacemnt.

An important fact must to be taken into account for the quadcopter model parameters introduced. Different batteries to the ones used in the manual missions were utilized due to larger battery life time was needed. These new batteries had larger mass (0.2kg) than the old ones, so the mass of the quadcopter changed. Also, since the maximum thrust that the rotors can develop is still the same, the hover throttle parameter need to change also.

$$T_{max} = \frac{m_1 g}{\Pi_{hov,1}} = \frac{m_2}{\Pi_{hov,2}} \tag{5.1}$$

So, new hover throttle value can be calculated from this equation.

$$\Pi_{hov,2} = \Pi_{hov,1} \frac{m_2}{m_1} \tag{5.2}$$

Finally, the following parameters were introduced in the default simulator model.

Mass (kg)	Hover Throttle (-)	Terminal Velocity (m/s)	Rotational terminal velocity (rad/s)
1.197	0.761356	12.5	8π

Table 5.2: Quadcopter parameters for the SITL dynamic equations - Automatic case

For checking if this result was coherent for the simulation, it was analyzed the throttle values for the real flight and the simulation. Similar throttle trends were obtained. Moreover, due to the fact that the battery is located at the centre of gravity of the quadcopter, there is no other change in any parameter.

Mission 1

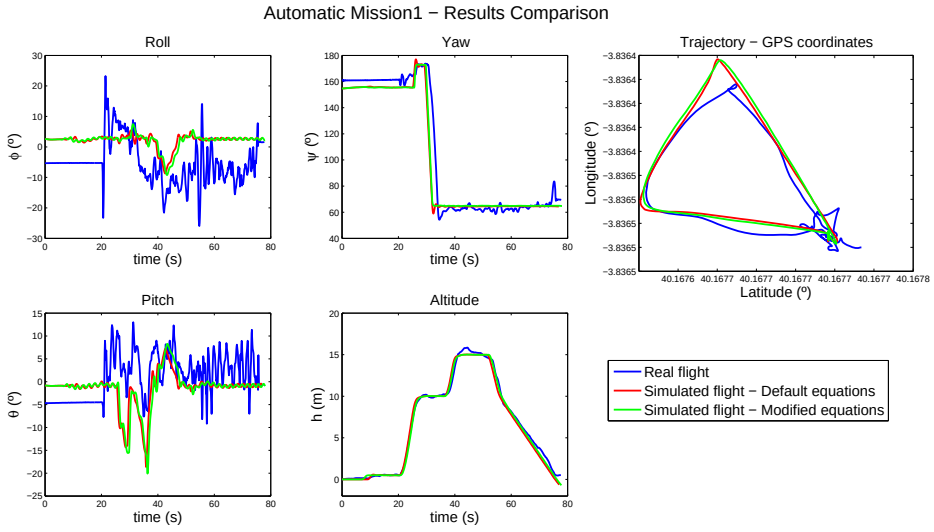

Figure 5.4: Automatic Mission 1 - Real vs Simulated flights data comparison.

It can be appreciated in Figure 5.4 that very similar results are obtained for both simulation cases. Furhtermore, the yaw together with the vehicle altitude and trajectory is succesfully simulated. In fact, the main conclusion is that the mission time predicted by the modified equations simulation case

for the automatic mission is equal to the one spent in reality. Therefore, it seems that the simulator can be used for estimating different mission times, which may be of primarly importance if it is wanted to adjust the battery time life without the necessity of doing a real test. The flying time is a bit smaller for the default equations simulation, since it lands a bit earlier than the real flight. Regarding the roll and pitch results, greater differences appears. The main flight peaks occurring at 45s from the take-off are well simlated. However, larger errors are found in the rest of measurements. It seems to be caused due to the large wind gusts occurring during the test, which also explains the sharp real values founded for the roll and the pitch.

Hence, it seems to be that in autopilot mode the influence of the PID controllers in the quadcopter dynamics is greater. Therefore, using an accurate description of the dynamic equations of the quadcopter has less influence in the final results. Nonetheless, the roll and pitch peaks are slightly better simulated by the case using modified equations, since larger values (which are closer to the real ones) are found in this case. Furthermore, it is remarkable that the simulated case with modified equations offers a better description of the mission, since it simulates closer to the reality the time that the vehicle needs to fulfill every mission requirement and to finish it.

Chapter 6

Conclusions and Future Work

In this chapter, the main conclusions reached are going to be presented. Also, the future work posibilities are going to be examined.

6.1 Conclusions

The main conclusions can be summarized in the following ones.

- Refinement of each quadcopter component, including vehicle calibration, lets to succesfully fly either manually and autonomous missions.
- A good characterization of the quadcopter physical properties leads to more realistic simulations.
- SITL is a suitable flight simulator for testing quadcopter dynamics, since it includes model for simulating the autopilot board performance. However, the quadcopter dynamics model implemented in this software is a simplification of the theoretical one.
- Manual flights simulations do not represent the real situation when using the SITL default equations, although a general overview of the quadcopter performance is obtained.
- The default SITL simulator offers an enough accurate prediction of quadcopter automatic flight. PID controllers effects are more present

in this kind of flight, so the influence of the rough dynamical model used is minor in this case.

- The improvement of the dynamic equations of motion used in SITL, consisting in adjusting the quadcopter parameter such as its moments of inertia or its arm length and adding other non-linear influences such as the gyroscopic effects, makes the simulation results to be more similar to the real ones. This can be especially appreciated in the case of manual control, where the orientation movements are accurately predicted. Regarding the automatic flight, the mission is slighly better simulated than the other case. Again, it seems to be due to the larger influence of the PID controllers in this kind of flights.

- Simulated results for the automatic missions are less rough than the real ones. It seems to be due to the fact that the real mission was affected with random gusts of wind, which produced impulse accelerations on the vehicle implying that larger angle oscillations were attained.

6.2 Future work

Possible future work have been studied, in order to be able to improve this project.

1. Using the simulator, once it is adjusted (the equations of motion have been modified), for testing quadcopter behaviour in different missions. Different tasks such the check that an automatic mission can be flown without ending the battery life time can be deduced by only using the simulator, since it predicts the mission time in a very accurate way.

2. Try to improve the simulation environment, looking for being able to predict the vehicle response when it is affected by constant wind or by sudden wind gusts.

3. Adding dynamic aeroelastic effects to the equations, in order to be able to simulate the quadcopter response to gusts.

4. In depth study of the sensor models used in SITL software. A more accurate modeling of them will help to improve the simulator capacity.

5. Test the behaviour of the simulator under new tests where larger speeds and altitudes are achieved by the quadcopter.

Bibliography

[1] J. Gundlach, *Designing unmanned aircraft systems: A comprehensive approach*. AIAA Education Series, 2012.

[2] R. P. Schwing, "Unmaned aerial vehicles - revolutionary tools in war and peace," 2007.

[3] L. Montesinos, "Aplicaciones de los drones en la agricultura de precisión," 2015.

[4] J. Camhi, "The drones report. market forecasts, key players and use cases, and regulatory barriers to the proliferation of drones," 2016.

[5] *Boletn Oficial del Estado*. No. 252, 2014.

[6] AESA, "`http://www.seguridadaerea.gob.es/lang_castellano/home.aspx`," 2016.

[7] L. Ribeiro and N. Oliveira, *UAV Autopilot Controllers Test Platform Using Matlab/Simulink and X-Plane*. Instituto Tecnológico de Aeronáutica, 2010.

[8] Y. C. Paw, "Synthesis and validation of flight control for UAV," Master's thesis, University of Minnesota, 2009.

[9] J. M. Brito, "Quadrotor prototype," 2009.

[10] S. Bouabdallah, P. Murrieri, and R. Siegwart, *Design and control of an indoor micro quadrotor*, vol. 5. IEEE, 2004.

[11] Ardupilot Mega Project, "`http://www.ardupilot.co.uk/`," 2016.

[12] 3D ROBOTICS, "`https://store.3dr.com/products/3dr-pixhawk`," 2016.

[13] S. Temizer, "The state of the art and the future of modeling and simulation," 2007.

[14] S. Bouabdallah, "Design and control of quadrotors with application to autonomous flying," Master's thesis, École Polytechnique federale de Lausanne, 2007.

[15] J. G. Leishman, *Principles of helicopter aerodynamics*. Cambridge University Press, 2006.

[16] D. de Bioingenierá e Ingeniería Aeroespacial, "Autonomous navigation of quadcopters," 2015.

[17] R. M. Murray, Z. Li, and S. S. Sastry, *A Mathematical Introduction to Robotic Manipulation*. CRC Press, 1994.

[18] M. A. Fernandez, "Assembly, modeling, simulation and control of a quadcopter for application to solar farm inspection," Master's thesis, Universidad Carlos III de Madrid, 2015.

[19] ArduPilot Dev Team, "http://ardupilot.org/dev/docs/sitl-native-on-windows.html," 2016.

[20] ArduPilot Dev Team, "http://ardupilot.org/dev/docs/sitl-simulator-software-in-the-loop.html," 2016.

[21] Dronecode Project, "http://dronecode.github.io/mavproxy/html/index.html," 2016.

[22] ArduPilot Dev Team, "http://ardupilot.org/planner/docs/mission-planner-overview.html," 2016.

[23] ArduPilot Dev Team, "http://ardupilot.org/dev/docs/apmcopter-code-overview.html," 2016.

[24] A. R. Galán, "Revisión y modificación del firmware de libre acceso arducopter para uso en el proyecto airwhale," Master's thesis, Universidad de Sevilla, 2015.

[25] F. Fernández Gutiérrez, "Diseño e implementación de un sitema de control asistido para plataforma aérea multi-rotor," Master's thesis, Universidad de Chile, 2015.

Appendix A

Data filter for manual flight logs

DataSelection.m

```
%% Useful data selection – From telemetry logs

clear all
close all
clc

mission=input('Introduce number of mission: \n');
IntPoint=input('Intersection point for altitude: \n (Values between [0 0.2]) ');

try
load(fullfile(strcat('Mission' ,num2str(mission),'.mat')));
catch err
error(strcat('no data found for mission ' ,num2str(mission)));
```

```
end

% Look for point where barometer reset --> Mission start
k=0;  %Control variables initialization
for i=1:length(CTUN(:,7))
    if k==0
        if CTUN(i,7)<IntPoint
            target=i;
            k=2;
        end
    end
end
if k~=2; %Error masage
    disp('Filter cannot be done.')
    disp('Try with a higher value for the intersection point')
else
else
    a=1e7; %Variable initialization
    b=1e7;
    for i=1:length(RCIN)
        x=abs(RCIN(i,2)-CTUN(target,2));
        y=abs(RCIN(i,2)-ATT(i,2));
        if a>x
            a=x;
            target2=i;
        end
        if b>y
            b=y;
            target3=i;
        end
```

```
    end

    % Create the trust channel from the point where the mission starts
    len=length(RCIN(:,3));
    RC1a=RCIN(target2:len,3);
    RC2a=RCIN(target2:len,4);
    RC3a=RCIN(target2:len,5);
    RC4a=RCIN(target2:len,6);

    chan1_raw_mavlink_rc_channels_raw_t_Filt=RC1a;
    chan2_raw_mavlink_rc_channels_raw_t_Filt=RC2a;
    chan3_raw_mavlink_rc_channels_raw_t_Filt=RC3a;
    chan4_raw_mavlink_rc_channels_raw_t_Filt=RC4a;

    len2=length(ATT);
    roll_mavlink_attitude_t_Filt=ATT(target3:len2,:);
    yaw_mavlink_attitude_t_Filt=ATT(target3:len2,:);
    pitch_mavlink_attitude_t_Filt=ATT(target3:len2,:);
    h=CTUN(target:length(CTUN),:);

    save(strcat('M' ,num2str(mission),'.mat'),...
    'chan1_raw_mavlink_rc_channels_raw_t_Filt',...
    'chan2_raw_mavlink_rc_channels_raw_t_Filt',...
    'chan3_raw_mavlink_rc_channels_raw_t_Filt',...
    'chan4_raw_mavlink_rc_channels_raw_t_Filt','h',...
    'roll_mavlink_attitude_t_Filt','yaw_mavlink_attitude_t_Filt',...
    'pitch_mavlink_attitude_t_Filt')
end
```

Appendix B

Radio control inputs obtention - Manual flight tests

Rcin_Creator.m

```
% Script to rewrite RC inputs from .mat file into a file with new format
% --> .txt
clear all
close all
clc

mission=input('Introduce number of mission: \n');
try
load(fullfile(strcat('M' ,num2str(mission),'.mat')));
catch err
error(strcat('no data found for mission ' ,num2str(mission)));
end
```

```
% Extraction of RC commands from logs into cell arrays
time_signal = chan1_raw_mavlink_rc_channels_raw_t_Filt(:,1);
RC1 ={chan1_raw_mavlink_rc_channels_raw_t_Filt(:,1),time_signal};
RC2 = {chan2_raw_mavlink_rc_channels_raw_t_Filt(:,1),time_signal};
RC3 = {chan3_raw_mavlink_rc_channels_raw_t_Filt(:,1),time_signal};
RC4 = {chan4_raw_mavlink_rc_channels_raw_t_Filt(:,1),time_signal};
RC(1,:) = RC1;
RC(2,:) = RC2;
RC(3,:) = RC3;
RC(4,:) = RC4;

% Writing formated RC inputs into new files for simulation tests

filename = strcat('Mission' ,num2str(mission),'.txt');
file = fopen(filename,'w'); %Ready for be written
for t = 1:length(RC{1,1})
    for c = 1:4
        fprintf(file,'rc %d %d\n',c,RC{c,1}(t));
    end
    fprintf(file,'sleep\n');
end
fclose(file);
```

Appendix C

Time vector for sending the Radio Control input commands

tdelayCreator.m

```
%% Time delay vector creation

mission=input('Introduce number of mission: \n');

load(strcat('Mission' ,num2str(mission),'.mat'))
load(strcat('M' ,num2str(mission),'.mat'))

lenFilt=length(chan1_raw_mavlink_rc_channels_raw_t_Filt);
lenReal=length(RCIN);

Timevector=RCIN(lenReal-lenFilt+1:end,2)-...
    RCIN(lenReal-lenFilt:end-1,2);
Timevector=Timevector*1e-3; %From ms to s
```

```
save(strcat('TimeVector' ,num2str(mission),'.mat'),'Timevector','-v7.3')
```

Appendix D

TimeVecudp.py

This python function is responsible of interacting with MavProxy simulator
for sending the radio control inputs at the right time.

```
# TimeVecudp.py
#
# uses RC log data from real flight to simulate it using SITL and MAVProxy
# this script uses the mavutil library to send commands to MAVProxy via a UDP port
#connection
# more info:
#http://copter.ardupilot.com/wiki/common-sitl-software-in-the-loopsimulator/

import sys, os
from pymavlink import mavutil
import time
import numpy as np, h5py # For loading .mat files (saved before in -v7.3 )

# utility function to send RC commands to SITL via connection named "udp_port",
# defined as a UDP port connection, in this case 14551, which must be
```

```
# indicated with the --out command in the string 'cmd' used by pexpect

def set_attitude(rc1, rc2, rc3, rc4):
    global udp_port
    values = [ 65535 ] * 8
    values[0] = rc1
    values[1] = rc2
    values[2] = rc3
    values[3] = rc4
    udp_port.mav.rc_channels_override_send(udp_port.target_system, \
          udp_port.target_component, *values)

# create a mavlink instance as a UDP port connection
udp_port = mavutil.mavlink_connection('udp:127.0.0.1:14550')
time.sleep(1)

# arm motors
udp_port.wait_heartbeat()
udp_port.set_mode('STABILIZE')
print('\nArmando motores')
time.sleep(1)

udp_port.arducopter_arm()
udp_port.motors_armed_wait()

# make sure we have GPS fix
udp_port.wait_gps_fix()

# file reading loop
log_rc = open('Mission7.txt','r')
```

```
f = h5py.File('TimeVector7.mat','r')
data = f.get('Timevector')
time_delay = np.array(data) # For converting to numpy array
line_count = 0
t_final = 60 #simulation time in seconds

# time counter
time_0 = time.time()
for line in log_rc:
    if (line.rstrip('\n')) == 'sleep':
            # here we must calculate how much time to wait to have a sleep time as
                #close as possible to $time_delay (because of processing time)
            end_time = start_time + time_delay[0][line_count]
            sleep_time = end_time - time.time()
            time.sleep(sleep_time)
            line_count = line_count + 1
    else:
        # we must identify the rc num and assign the data
        [txt,num,val] = line.split(' ')
        if int(num) == 1:
            # reset time for the wait loop
            start_time = time.time()
            rc1 = int(val)
        elif int(num) == 2:
            rc2 = int(val)

        elif int(num) == 3:
            rc3 = int(val)
        elif int(num) == 4:
            rc4 = int(val)
```

```
            # when we have all 4 PWM values for the rc, we send it via udp
            set_attitude(rc1, rc2, rc3, rc4)
    # uncomment these 2 lines to stop simulation at $t_final
    #if line_count>= t_final*freq:
    # break
    print('Tiempo total: ' + str(time.time()-time_0))

# close file

log_rc.close()
```

Appendix E

Manual flights data comparison script

GlobalComparisonManual.m

```
%% Comparison data real vs simulated
clear all

mission=input('Introduce number of mission: \n');
lognumber=input('Introduce log number (base equations simulation): \n');
lognumber2=input('Introduce log number (modified equations simulation): \n');

load(strcat('M' ,num2str(mission),'.mat'))
file=fullfile('C:\cygwin\home\Doble\ardupilot\ArduCopter\logs'...
,strcat(num2str(lognumber),'.mat'));
load(file)

timesRS %Call this script
```

```
% Initial conditions for simulation

ICattVariation=yaw_mavlink_attitude_t_Filt(1,8)-ATT(1,8);
 %Establishing same initial yaw
for i=1:length(ATT)
    ATT(i,8)=ATT(i,8)+ICattVariation;
    if ATT(i,8)>360
        ATT(i,8)=ATT(i,8)-360;
    end
    if ATT(i,8)<0
        ATT(i,8)=ATT(i,8)+360;
    end
end
tSposbs=tSpos; %Renaming variables
tSbs=tS;
tSattbs=tSatt;
POSbs=POS;
GPSbs=GPS;
ATTbs=ATT;
GPSbs(:,11)=GPSbs(:,11)-GPSbs(1,11);

% Modified equations
file=fullfile('C:\cygwin\home\Doble\ardupilot\ArduCopter\logs'...
,strcat(num2str(lognumber2),'.mat'));
load(file)

timesRS1

ICattVariation=yaw_mavlink_attitude_t_Filt(1,8)-ATT(1,8);
```

```
%Establishing same initial yaw
for i=1:length(ATT)
    ATT(i,8)=ATT(i,8)+ICattVariation;
    if ATT(i,8)>360
        ATT(i,8)=ATT(i,8)-360;
    end
    if ATT(i,8)<0
        ATT(i,8)=ATT(i,8)+360;
    end
end
GPS(:,11)=GPS(:,11)-GPS(1,11);

% Graph
figure()
subplot(2,2,1)
plot(tRat,roll_mavlink_attitude_t_Filt(:,4),'b')
hold on
plot(tSattbs,ATTbs(:,4),'r')
hold on
plot(tSatt,ATT(:,4),'g')
xlabel('time (s)')
ylabel('\phi ( )')
legend('Real flight','Simulated flight - Default equations',...
'Simulated flight - Modified equations')
title('Roll')

subplot(2,2,2)
plot(tRat,yaw_mavlink_attitude_t_Filt(:,8),'b')
hold on
plot(tSattbs,ATTbs(:,8),'r')
```

```
hold on
plot(tSatt,ATT(:,8),'g')
xlabel('time (s)')
ylabel('\psi ( )')
legend('Real flight','Simulated flight - Default equations',...
'Simulated flight - Modified equations')
title('Yaw')

subplot(2,2,3)
plot(tRat,pitch_mavlink_attitude_t_Filt(:,6),'b')
hold on
plot(tSattbs,ATTbs(:,6),'r')
hold on
plot(tSatt,ATT(:,6),'g')
xlabel('time (s)')
ylabel('\theta ( )')
legend('Real flight','Simulated flight - Default equations',...
'Simulated flight - Modified equations')
title('Pitch')

subplot(2,2,4)
plot(tRh,h(:,7),'b')
hold on
plot(tSposbs,GPSbs(:,11),'r')
hold on
plot(tSpos,GPS(:,11),'g')
xlabel('time (s)')
ylabel('h (m)')
title('Altitude')
legend('Real flight','Simulated flight - Default equations',...
```

```
'Simulated flight – Modified equations')
suptitle(strcat('Mission ' ,num2str(mission),...
' – Real vs Simulated flight data'))
```

timeRS.m

```
%Time for real flight - RC
load(strcat('TimeVector',num2str(mission),'.mat'))

for i=1:length(Timevector)
if i==1
tR(i)=0;
else
tR(i)=tR(i-1)+Timevector(i);
end
end
numat=length(yaw_mavlink_attitude_t_Filt);
numh=length(h);
tfin=tR(end);
averat=tfin/numat;
averh=tfin/numh;
for i=1:numat
    if i==1
        tRat(i)=0;
    else
        tRat(i)=averat+tRat(i-1);
    end
end
for i=1:numh
    if i==1
    tRh(i)=0;
    else
    tRh(i)=averh+tRh(i-1);
    end
```

```
end

% Time for simulated flight - RC
for i=1:length(RCIN)
tS(i)=1e-6*(RCIN(i,2)-RCIN(1,2));
end
avv=tfin/length(ATT);
for i=1:length(ATT)
if i==1
        tSatt(i)=0;
    else
        tSatt(i)=avv+tSatt(i-1);
    end
end
for i=1:length(GPS)
tSpos(i)=1e-6*(GPS(i,2)-GPS(1,2));
end
```

timeRS1.m

```
% Time for simulated flight - RC - For GlobalComparisonManual.m Code
for i=1:length(RCIN)
tS(i)=1e-6*(RCIN(i,2)-RCIN(1,2));
end
% for i=1:length(ATT)
% tSatt(i)=1e-6*(ATT(i,2)-ATT(1,2));
% end
avv=tfin/length(ATT);
tSatt=0; %Reinitialize variable
tSpos=0;
```

```
for i=1:length(ATT)
if i==1
        tSatt(i)=0;
    else
        tSatt(i)=avv+tSatt(i-1);
    end
end
for i=1:length(GPS)
tSpos(i)=1e-6*(GPS(i,2)-GPS(1,2));
end
```

Appendix F

Autonomous flight data comparison

GlobalComparisonAuto.m

```
%% Comparison data real vs simulated for Automatic flight
clear all

mission=input('Introduce number of mission: \n');
lognumber=input('Introduce log number (base equations simulation): \n');
lognumber2=input('Introduce log number (modify equations simulation): \n');

%% Real mission data
load(strcat('Mission' ,num2str(mission),'Auto.mat'))
ATTReal=ATT;
GPSReal=GPS;
% Time for real mission
tATT=ATT(:,2)*1e-3; %from ms to s
```

```
tGPS=GPS(:,14)*1e-3; %from ms to s
% For time start from 0
t1ATT=tATT(1);
t1GPS=tGPS(1);
for i=1:length(tATT)
    tATTReal(i)=tATT(i)-t1ATT;
end
for i=1:length(tGPS)
    tGPSReal(i)=tGPS(i)-t1GPS;
end

%% Simulated data Base eq
file=fullfile('C:\cygwin\home\Doble\ardupilot\ArduCopter\logs'...
,strcat(num2str(lognumber),'.mat'));
load(file)
ATTbs=ATT; %bs-->Base equations simulation
GPSbs=GPS;
% time for simulation
tATTbs=ATT(:,2)*1e-6; %from ms to s
tGPSbs=GPS(:,2)*1e-6; %from ms to s
% For time start from 0
t1ATTbs=tATTbs(1);
t1GPSbs=tGPSbs(1);
for i=1:length(tGPSbs)
    tGPSbs(i)=tGPSbs(i)-t1GPSbs;
end
for i=1:length(tATTbs)
    tATTbs(i)=tATTbs(i)-t1ATTbs;
end
```

```
%% Simulated data Modified eq
file=fullfile('C:\cygwin\home\Doble\ardupilot\ArduCopter\logs'...
,strcat(num2str(lognumber2),'.mat'));
load(file)
% time for simulation
tATT=ATT(:,2)*1e-6; %from ms to s
tGPS=GPS(:,2)*1e-6; %from ms to s
% For time start from 0
t1ATT=tATT(1);
t1GPS=tGPS(1);
for i=1:length(tGPS)
    tGPS(i)=tGPS(i)-t1GPS;
end
t1ATT=tATT(1);
for i=1:length(tATT)
    tATT(i)=tATT(i)-t1ATT;
end

%% Graph
figure()
subplot(2,3,1)  %Roll
plot(tATTReal,ATTReal(:,4),'b')
hold on
plot(tATTbs,ATTbs(:,4),'r')
hold on
plot(tATT,ATT(:,4),'g')
xlabel('time (s)')
ylabel('\phi ( )')
legend('Real flight','Simulated flight - Default equations',...
'Simulated flight - Modified equations')
```

```
title('Roll')



subplot(2,3,2)
plot(tATTReal,ATTReal(:,8),'b') %Yaw
hold on
plot(tATTbs,ATTbs(:,8),'r')
hold on
plot(tATT,ATT(:,8),'g')
xlabel('time (s)')
ylabel('\psi ( )')
legend('Real flight','Simulated flight - Default equations',...
'Simulated flight - Modified equations')
title('Yaw')

subplot(2,3,4)
plot(tATTReal,ATTReal(:,6),'b')  %Pitch
hold on
plot(tATTbs,ATTbs(:,6),'r')
hold on
plot(tATT,ATT(:,6),'g')
xlabel('time (s)')
ylabel('\theta ( )')
legend('Real flight','Simulated flight - Default equations',...
'Simulated flight - Modified equations')
title('Pitch')

subplot(2,3,5)
plot(tGPSReal,GPSReal(:,9),'b')  %Rel Alt
hold on
```

```
plot(tGPSbs,GPSbs(:,10),'r')
hold on
plot(tGPS,GPS(:,10),'g')
xlabel('time (s)')
ylabel('h (m)')
legend('Real flight','Simulated flight – Default equations',...
'Simulated flight – Modified equations')
title('Altitude')

subplot(2,3,3)
plot(GPSReal(:,7),GPSReal(:,8),'b')  %Latitude & Longitude
hold on
plot(GPSbs(:,8),GPSbs(:,9),'r')
hold on
plot(GPS(:,8),GPS(:,9),'g')
xlabel('Latitude ( )')
ylabel('Longitude ( )')
legend('Real flight','Simulated flight – Default equations',...
'Simulated flight – Modified equations')
title('Trajectory – GPS coordinates')
suptitle(strcat('Automatic Mission ' ,num2str(mission),' – Results Comparison'))
```

Appendix G

Modified equations of motion in SITL

In this appendix it is presented the modified equations of motion written in C++. It is a part of the SIM_Multicopter.cpp file, and it is a modifications of the lines going from 142 to 185.

```
// calculate rotational and linear accelerations
void Frame::calculate_forces(const Aircraft &aircraft,
                             const Aircraft::sitl_input &input,
                             Vector3f &rot_accel,
                             Vector3f &body_accel)
{
    // rotational acceleration, in rad/s/s, in body frame
    float thrust = 0.0f; // newtons

    for (uint8_t i=0; i<num_motors; i++) {
        float motor_speed = constrain_float(
            (input.servos[motor_offset+motors[i].servo]-1000)/1000.0, 0, 1);
```

```
        rot_accel.x  += -0.16593*thrust_scale/0.014201403 *
            sinf(radians(motors[i].angle)) * motor_speed;
        rot_accel.y  +=  0.16593*thrust_scale /0.013722055 *
             cosf(radians(motors[i].angle)) * motor_speed;
        rot_accel.z += motors[i].yaw_factor * motor_speed *
            0.16593*thrust_scale/0.019468594;
        thrust += motor_speed * thrust_scale; // newtons
    }

    body_accel = Vector3f(0, 0, -thrust / mass);

    if (terminal_rotation_rate > 0) {
        // rotational air resistance
        const Vector3f &gyro = aircraft.get_gyro();
        rot_accel.x -= gyro.x * gyro.x * 0.16593*thrust_scale/0.014201403/
             terminal_rotation_rate /terminal_rotation_rate;
        rot_accel.y -= gyro.y * gyro.y * 0.16593*thrust_scale /0.013722055/
             terminal_rotation_rate /terminal_rotation_rate;
        rot_accel.z -= gyro.z * gyro.z * 0.16593*thrust_scale/0.019468594/
             terminal_rotation_rate /terminal_rotation_rate;
        // body gyroscopic effects
        rot_accel.x += -gyro.y*gyro.z*(0.013722055-0.019468594)/0.014201403;
        rot_accel.y += gyro.x*gyro.z*(0.019468594-0.014201403)/0.013722055;
        rot_accel.z += gyro.x*gyro.y*(0.014201403-0.013722055)/0.019468594;
    }

    if (terminal_velocity > 0) {
        // air resistance
        Vector3f air_resistance = -aircraft.get_velocity_ef() *
             (GRAVITY_MSS/terminal_velocity);
```

```
        body_accel += aircraft.get_dcm().transposed() * air_resistance;
    }

    // add some noise
    const float gyro_noise = radians(0.1);
    const float accel_noise = 0.3;
    const float noise_scale = thrust / (thrust_scale * num_motors);
    rot_accel += Vector3f(aircraft.rand_normal(0, 1),
                          aircraft.rand_normal(0, 1),
                          aircraft.rand_normal(0, 1)) * gyro_noise * noise_scale;
    body_accel += Vector3f(aircraft.rand_normal(0, 1),
                           aircraft.rand_normal(0, 1),
                           aircraft.rand_normal(0, 1)) * accel_noise * noise_scale;
}
```

The numerical values corresonds to the quadcopter properties.

Arm length $l(m)$	$I_{xx}(kgm^2)$	$I_{yy}(kgm^2)$	$I_{zz}(kgm^2)$
0.16593	0.014201403	0.013722055	0.019468594

Table G.1: Quadcopter arm length and moments of inertia.